FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Industrial Mobile App for a Sensor Cloud

João Carlos Barreira Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Gil Manuel Gonçalves

Co-Supervisors: João Reis and Susana Aguiar

July 24, 2017

Industrial Mobile App for a Sensor Cloud

João Carlos Barreira Fernandes

Mestrado Integrado em Engenharia Informática e Computação

July 24, 2017

Resumo

Ao longo dos últimos anos, o conceito de *Intelligent Manufacturing*, tem sido alvo de atenção por grande parte das empresas do meio. A virtualização do chão de fábrica, o rápido e fácil acesso a informação sobre as máquinas e a colaboração entre equipamentos no fabrico, são alguns exemplos das últimas tendências nesta área. Estes desenvolvimentos ao nível do *software*, *hardware* e tratamento de dados podem ter um impacto muito significativo no modo como o *shop-floor* é compreendido, analizado, interpretado e operado.

Neste contexto foi desenvolvido o *SelSus - Health Monitoring and Life-Long Capability Management for SELF-SUStaining Manufacturing Systems*. Este projeto surge como uma ferramenta de diagnóstico integrada, com conhecimento do estado e do histórico de todos os componentes que constituem um sistema ou uma fábrica, através da *Sensor Cloud* e da *Wireless Sensor Network*.

Atualmente os *smartphones* são um dos dispositivos electrónicos mais utilizados. Estes possuem diversos sensores e funcionalidades embutidas que podem ser aplicadas na indústria para supervisionar máquinas, operadores e o processo de interação entre estes e o sistema/processo de fabrico. Com esta ideia em mente, o foco principal deste projecto é o desenvolvimento de uma aplicação móvel, em Android, que permita a captura e monitorização de informação importante relacionada com a máquina e com o estado do sistema. Esta aplicação tem duas funcionalidades principais: uma interface para mostrar a informação e o processamento embebido nas máquinas, quer em tempo real, quer para um determinado período de tempo, e uma ferramenta que permite ao utilizador capturar informação dos vários sensores integrados com o *smartphone*. O equipamento será aproveitado ao máximo em termos da quantidade de dados que pode ser adequirida.

O principal objectivo da industria prende-se com a redução de custos, o que pode ser atingido reduzindo o tempo de inactividade das máquinas. Em caso de falha, a linha de produção em que a máquina está integrada pára, o que representa um desperdício de tempo de produção e pode também provocar danos nas matérias primas. A aplicação desenvolvida torna possível recolher dados que podem ser utilizados pelas técnicas de manutenção preventiva. Isto é possível, uma vez que toda a informação é enviada para a sensor cloud e tratada como informação adicional da máquina. O objectivo principal é então, monitorizar o estado da máquina e tentar prever falhas para que a manutenção adequada possa ser efectuada de ante-mão, evitando perda de desempenho.

Palavras-chave: Intelligent Manufacturing; Sensor Cloud; Wireless Sensor Network; Smartphone; Industry 4.0; SelSus.

Abstract

In the field of Intelligent Manufacturing, some topics have gotten a great deal of attention in the past few years.. The virtualization of shop-floor equipment, the easy access to the machine information or the collaboration among shop-floor equipment are examples of the latest trends in this field. This technological developments in software, hardware and data treatment can have a huge impact on how the shop-floor is perceived, analyzed, interpreted and operated.

In this context, the SelSus - Health Monitoring and Life-Long Capability Management for SELF-SUStaining Manufacturing Systems project was developed. This project comes as a new synergetic diagnostic and prognosis environment fully aware of the condition and history of all machine components within a system or factory, making use of a Sensor Cloud and a Wireless Sensor Network.

Presently, smartphones are one of the most used electronic devices and they have many sensors and built in functionalities that can be used in the industry for monitoring the process of machines and operators' interaction with the manufacturing system. Following this idea, the main focus of this thesis is the development of an Android mobile application that allows the capture and monitoring of important information related to the machines and system state. This application has two main functionalities: an interface to show the machine related data and analysis, both in real-time and non-real time, and a tool that allows the user to capture data from the various sensors built in the smartphone. This equipment will be used to its fullest in terms of how much data it can acquire.

The main goal in the industry is to reduce costs, which can be achieved by reducing the down time of the machines. If the machine fails, the production line in which that machine is integrated stops, which results in a waste of production time and may also damage raw materials. The developed application can be used to aid predictive maintenance applications as well as aid in the prevention of unexpected failures in the industrial equipments. This can be achieved, as all data will be sent to the Sensor Cloud and treated as additional information for a specific machine. The main objective is to check the state of the machine and try to predict failures so that maintenance can be performed beforehand, avoiding its loss of performance.

Keywords: Intelligent Manufacturing; Sensor Cloud; Wireless Sensor Network; Smartphone; Industry 4.0; SelSus; Machine Monitoring; Machine Analysis; Predictive Maintenance.

Acknowledgements

First of all I would like to thank my parents and relatives for their guidance and unconditional support through the last years, without them none of this would have been possible.

Secondly, I would like to express my feelings of great respect and appreciation towards my supervisors, Professor Gil Gonçalves, João Reis e Susana Aguiar for their endless support through the development of this dissertation.

Thirdly, I would like to thank all my friends and colleagues, for their assistance and companionship throughout the years.

Lastly, I want to express my deepest gratitude towards FEUP and all my professors for providing me with the best tools to succeed.

João Fernandes

"The best way to predict the future is to invent it."

Alan Kay

Contents

List of Figures

List of Tables

Abbreviations

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CoAP	Constrained Application Protocol
CPS	Cyberphysical Systems
CPPS	Cyberphysical Production Systems
DDS	Data Distribution Service
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IIoT	Industrial Internet of Things
IoT	Internet of Things
JMS	Java Message Service
JSON	JavaScript Object Notation
ME	Maintenance Engineer
MO	Machine Operator
M2M	Machine to machine
Mo2M	Mobile to machine
MQTT	Message Queuing Telemetry Transport
NFC	Near Field Communication
QR code	Quick Response code
REST	Representational State Transfer
RFID	Radio-frequency identification
SC	Sensor Cloud
SDK	Software Development Kit
SSD	Selcomp Self-description
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
XML	eXtensible Markup Language
WSN	Wireless Sensor Networks

Chapter 1

Introduction

We are in an era where production processes are practically fully automated. Manufacturing enterprises need to adopt new technologies to advance in a time where flexibility, scalability and agility are the differentiating factors [MMB16], where the entire production chain, including suppliers, logistics and product life cycle management will be connected, in order to optimize the production process and improve its efficiency.

The fast development of the wireless networks entailed the appearance of Wireless Sensor Networks (WSN). These consist in multiple small, low-power and low-price sensor nodes able to collect, store, and process environmental information as well as communicate with other nodes around them [HT04]. The huge amount of sensor data being generated and the necessity to access and analyze it at different times and locations led to the appearance of a new type of service architecture named Sensor Cloud (SC), an integration of cloud computing into the WSN. This architecture uses virtual sensors on top of the physical wireless sensors that users automatically and dynamically can utilize through different applications [MP16].

In this first chapter will present the context, goals and motivation for this work, alongside with the structure of the thesis.

1.1 Context

In the industry field, the Internet of Things connects sensor networks and machines to the internet allowing them to communicate and share its data. Even the simpler devices are able to send data through the network allowing the whole system to access it in real-time. The cloud is able to process this information, make some decisions and take actions.

In the 18th century, with the first industrial revolution, hand production methods were replaced by machines powered by water and steam. Later in the 19th century electricity and the conveyor belt brought the possibility of mass production and that was the second technological revolution [acc]. The appearance of computers and automation processes was in the basis of the third technological revolution. Nowadays we are experiencing the fourth revolution or the Industry 4.0, where the Internet of Things, the rise of cyber-physical systems and the cloud systems originated

the Smart Factories. This cyber-physical systems are able to monitor physical processes, create a virtual copy of the physical world and make decentralized decisions [ind].

This dissertation is framed on the SelSus project [Sel], Health Monitoring and Life-Long Capability Management for SELf-SUStaining Manufacturing Systems, an European collaboration project where the Institute for Systems and Robotics from FEUP participates. Its work aims on improving the production processes by developing self-healing production systems that can live longer by using a predictive maintenance and constantly being renovating and upgrading.

1.2 Motivation

SelSus [Sel] and other related projects work to help the European manufacturing industry to grow, by developing innovative concepts for more maintainable and flexible shop-floors. This solutions aim to merge both the real and virtual worlds by connecting all physical devices allowing them to share data and be aware of its surroundings increasing its visibility and flexibility for data handling and access in manufacturing systems [RPG17]. These developments bring important costs reduction that aid with the European economy sustainability.

This thesis aims on reducing the time spent on machine maintenance by providing a simple and portable way to access machine data and reduce the time spent on diagnosis. This is achieved by stepping up at different phases of the machine's life-cycle: The machine integration and ramp-up; The installation and configuration of new elements; The maintenance of machine components. In figure 1.1 we can see a comparison between the conventional integration and ramp-up approach and SelSus', through different phases.

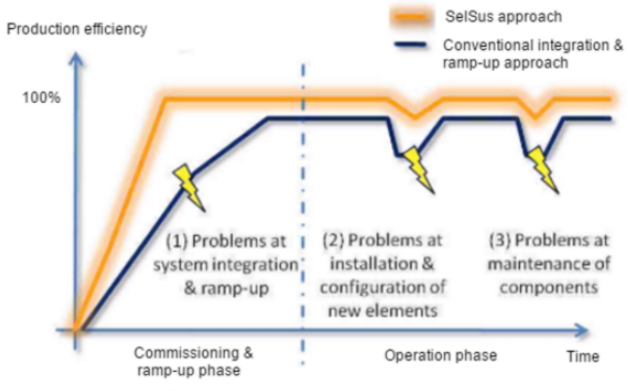

Figure 1.1: Selsus expected production efficiency

Another motivational aspect is the fact that the smartphone is one of the most used mobile devices, constantly being upgraded either by new software updates or the release of new models with more hardware capabilities such as better processors or new sensors.

1.3 Objectives

In this project we present an Android mobile application able to connect with the SelSus environment as a new smart component capable of receive and generate data from a set of internal and external sensors. This data can later be processed by the system in order to change its behaviour or to calibrate a specific sensor.

This application allow users to visualize information from a specific machine both in real-time and from the machine history. Making this a simple and portable way to overview the state of the system from anywhere. This features can help operators to make easier and faster diagnosis and maintenance, over the shop-floor equipments.

With the real-time visualization of all data being generated by the machine embedded sensors and the possibility to directly calibrate this device introduces the Mobile to Machine (Mo2M) concept. This model makes use of the latest advances in the IoT technologies to fully integrate the smartphone into the smart factories.

The main objectives behind this dissertation's work are: Assist the machine operator and the maintenance engineer at their diagnosis and maintenance operations over shop-floor machines; Create a Mobile to Machine (Mo2M) concept where the smartphone is able to access the real-time data being generated by shop-floor machines and these are prepared to receive data from the smartphone embedded sensors, process it and run recalibration processes based on it; Verify the utility of the developed application in the production manufacturing context.

1.4 Methodology

This dissertation's work is marked by different phases. At first, some functional and nonfunctional requirements where analyzed so we could comprehend all the functionalities our application should meet. With a better understanding on the problem we took some time to review the related literature and to search for the technologies that would better fit our project's constraints.

With all requirements set, we started the implementation phase and during this some requirements were re-adjusted during several team meeting that took place. After finishing the implementation the application was integrated in the Selsus environment and some validation tests were performed.

The described working process follows the Waterfall model, which divides the project in defined phases that occur in a linear and sequential order. In the following figure we present our development methodology different steps.

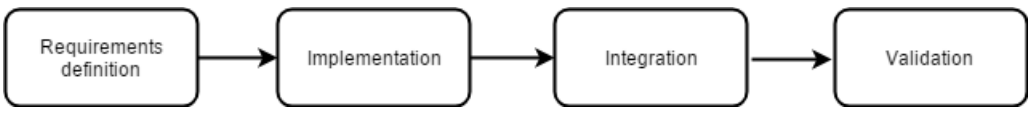

Figure 1.2: Development methodology diagram.

1.5 Document Structure

Besides this introductory chapter, this document contains 5 more, Chapter 2 contains the state of the art, some projects that are in the basis of this dissertation and some technologies to be used in the application development. Chapter 3 is about the Selsus, where we present more details about the project, its architecture, the different components and specifications. Chapter 4 describes the proposed solution, including its architecture and details about the implementation. After this in Chapter 5 we present the validation tests for our application and finally in chapter 6 we have the conclusions for this project and some future work.

Chapter 2

Related work and state-of-the-art

In this chapter we present the literature review concerning the problem at hand. First, some of the most relevant topics related to the dissertation theme are introduced. The Internet of Thing (IoT) and the Industrial Internet of Things (IIoT) are growing with new products being developed every day.

After this, some related European projects are presented as well as a study on the technologies taken into account for the development of this thesis. These technologies include the smartphone and its embedded sensors used in the implementation phase. At the end we present some related projects also working on the shop-floor smartphone integration.

2.1 Internet of Things, Industrial Internet of Things & Cyber-Physical systems

In the last few years the term "Internet of things" (IoT) has became widely popular. This concept is a new paradigm where all physical machines can be equipped with identifying, sensing, networking and processing capabilities [WADX15]. Several consortia have been formed to develop frameworks and standards for the IoT and companies started working on IoT based products and services to meet the requirements of a more and more competitive industry.

In this context we are introduced to the Industry 4.0, a vision for a more advanced production system control architecture and engineering methodology. This concept was introduced by the German federal government in 2011 but it's frequently associated with the 4th Industrial Revolution. The first three revolutions came as a result of technical innovations: the introduction of water and steam-powered mechanical manufacturing systems at the end of the 18th century who came to replace some of the human and animal work, the electricity appearance and mass-production systems at the beginning of the 20th century and the automation of the manufacturing processes at the end of the 20th century [Ind13] [BFKR14].

While Industry 3.0 focused on the automation of single machines and processes, Industry 4.0 focus is on the virtualization of all physical assets and integration into a digital ecosystem [RJS16]. This allows a better understanding on the overall system and its components, since

all data generated by all physical devices can be accessible and processed towards this system's optimization.

With this technological revolution comes an increased integration of Cyber-Physical Systems (CPS), where computational entities are connected with the surrounding physical world allowing the use of data-access and data-process applications available over the network [Mon14]. The research about CPS is growing by identifying opportunities, challenges, and needs in any industrial sector and from European Union's investment in multidisciplinary research teams to work in this new technologies. [BG11].

CPS are implemented in all kinds of sectors, such as autonomous cars, medical monitoring systems and domotics. When applied to the manufacturing industry, this model is known as Cyber-Physical Production System (CPPS). This concept is based on the fusion of CPS, Service-Oriented Architectures (SOA), Cloud and Fog Computing and it is the outcome of the communication between different elements across all levels of production, changing its operations and behaviour according to the state of the whole production system [RPG17] [Mon14].

2.2 European industrial projects

As mentioned earlier in section 1.1 this dissertation is framed on the Selsus project scope. Before this, the Institute for Systems and Robotics participated in other European projects, which had similar field of studies focus and where the developed models are the foundations for the Selsus project. In these projects were developed different components and solution that are part of the SelSus architecture and environment. In this section an overview of those projects will be provided.

2.2.1 XPRESS

XPRESS [XPR] was one of the big collaborative projects of the European Union in the area of flexible production. Running from 2007 to 2011, this project aimed to establish a breakthrough for the smart factory field, developing a new flexible production concept based on the idea of specialized intelligent process units, named expertons, integrated in cross-sectoral learning networks for a customized production. This concept was used in the automotive, aeronautics and electrical industry but can be applied to other industry production processes since it is flexible, in term of the integration of new modules and functionalities.

Expertons are a combination of different classical hardware and software components of assembly processes with an advanced knowledge system. What gives them the ability to choose the best known production parameters for a given task. This component also has the capability to learn the best practises both by the use of self-learning algorithms and data mining. This design would allow to respond to rapidly changing consumer needs, producing large quantities of high quality products at a low cost.

2.2.2 I-RAMP [3]

I-RAMP[3] [IRA], or Intelligent Reconfigurable Machines for Smart Plug&Produce Production, is an European collaborative project that involves both industrial and academic partners from Germany, Portugal, Netherlands, Hungary, France and Greece and it pretends to change conventional production equipment into Network-enabled Devices (NETDEVs). Those devices are controlled by virtual agents and have standardized interfaces and communication protocols that allow collaboration and negotiation between others being able to adopt themselves to varying production setup and production conditions [IRA]. NETDEVs can both be physical or logical devices that are components from a shop floor virtualization.

Sensor data is extremely important when it comes to monitoring machines or its environmental surroundings, machine diagnosis or process adaptation to new requirements [PRS+16].This technologies allows Wireless Sensor Networks (WSNs) to become more flexible and agile enhancing shop floor operations [PRS+15].

2.2.3 ReBorn

During its life cycle manufacturing equipment passes through different stages, from its assembly on the production line to its end-of-use. In some stages failures can potentially cause costly machine downtime or even downtime in the entire production system.

The ReBorn project [ReB] aims on demonstrate strategies and technologies that support the reuse of decommissioned production systems and equipment, in a way that they can be adapted to a different production design. This is achieved with a potential change, upgrade, reuse or dismantle of the stalled equipment making it useful possibly in a new context. This new strategies will contribute to sustainable, resource-friendly and green manufacturing and, at the same time, deliver economic and competitive advantages for the manufacturing sector [Fer15] [APG16] [FAPG16].

2.3 Smartphones, sensors and applications

Nowadays smartphones are one of the most used mobile devices, until the end of 2017 the number of smartphone users should reach 2.3 billions worldwide [stab]. As we can see at the statista study results (figure 2.1) these numbers are growing with more and more people start using this type of equipment everyday. Smartphones are now viewed as pocket computers rather than as phones due to their processing power, storage, large screens and open operative systems that encourage people to develop new applications.

Smartphones capabilities combined with its portability allow its users to have different kinds of applications and to access a large variety of contents everywhere they wish. At the shop-floor operators need to be aware of machines condition and, in some cases, the interfaces for this information are far from both the user and the machine making this diagnosis and overview actions more time consumer. The integration of the smartphone in this system can bring those interfaces closer to the operator.

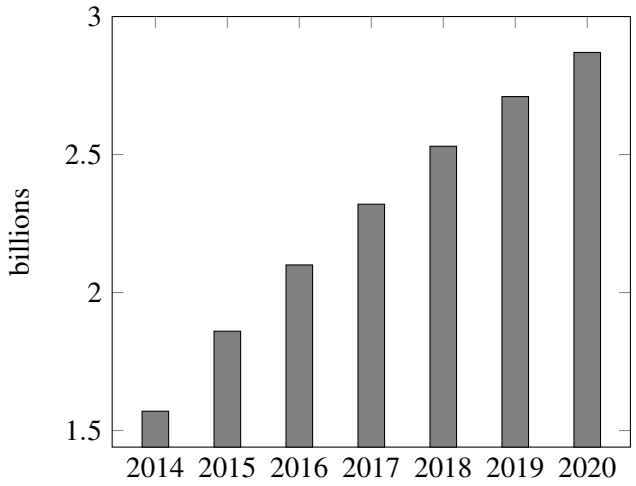

Figure 2.1: Number of smartphone users worldwide [stab]

The number of smartphone embedded sensors has grown in the last few years, each new smartphone model has more and better built-in sensors to support new features and the development of more precise applications. These sensors started to help on simple tasks like the use accelerometer to calculate the screen orientation but now they are used in lots of different applications [LML+10].

There are some companies who developed external sensors able to connect with android, those can have either wired or wireless connected to the smartphone using it as an interface to display its data. Most of this products come with its own mobile application to work with them and are not open for other developer use it in their own applications. Some of this sensors will be described in the following sections.

2.3.1 Internal sensors

In this subsection, the most commonly used smartphone internal sensors will be described. Among these we have orientation sensors such as the accelerometer, the gyroscope, and the magnetometer; proximity sensors; light sensors; environment sensors such as barometer, thermometer, and air humidity.

The accelerometer measures the proper acceleration that the handset is experiencing relative to free fall, also used to determine a device's orientation along its three axis. The gyroscope also provides orientation information but with more precision, it is used to capture rotation values, making possible to know the specific position in space of the smartphone. The last of these orientation sensors is the magnetometer, it can detect magnetic fields allowing compass applications to work.

Smartphones have a proximity sensor able to measure the distance from the closest object in range. It is used to turn off the screen during calls or to detect if the phone is inside the pocket. Light sensor measure the ambient light level and it is commonly used to auto correct the display brightness. About the environment conditions we have the barometer and thermometer used to measure the atmospheric pressure and the ambient temperature respectively and the air humidity

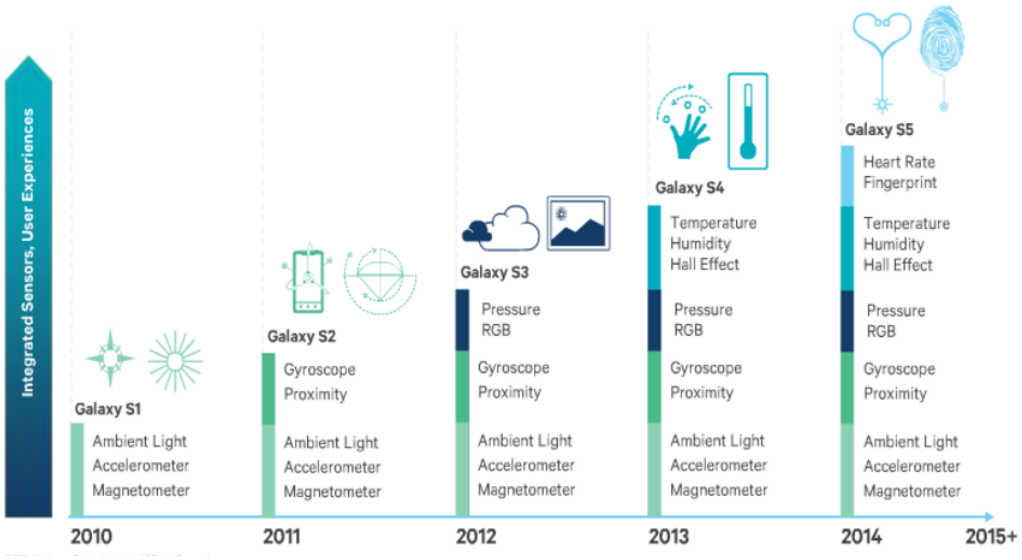

Figure 2.2: Samsung Galaxy S sensor growth [quab].

sensor. These equipments also have a built-in camera and microphone used to images, sound and video capture. The figure 2.2, provided by Qualcomm [quaa], presents the Samsung [sam] model S embedded sensor growth in its earlier models (S1 to S5). This smartphone is at the moment in its 8th generation, which has all sensors from the S5 model plus a barometer and a biometric sensor for iris recognition.

The number and type of sensors available in a smartphone depend on its model. The Android sensor framework [senb] helps dealing with all the different sensor manufactures and models, providing methods to operate each sensor. In the table 2.1 we list the sensors available at the android framework and the API versions supporting them.

2.3.2 External sensors

In the search for external sensors to integrate with our application we found that the Android sensor manager doesn't support external devices. Some companies already presented solutions for the use of external sensors with the smartphone and specifically with Android, providing its own applications to interact with the sensor data.

Since we need to integrate these sensors with our application we need a solution that provides a specification for the communication between the application and the sensor. Most of the solutions we found were developed aiming its use in research labs or domotics and not for the manufacturing industry.

These sensors can exist in a standalone context or as part of a more complex device with different embedded sensors. These can be connected via USB, directly to the smartphone or via a proper interface device able to translate the sensor data, and wirelessly via Bluetooth.

Vernier [ver] offers a large variety of sensors that can be used with graph applications they provide, although they are planning on it, these sensors doesn't have an SDK for its integration in

Table 2.1: Sensor availability by platform.

Sensors	**Android 4.0 (API Level 14)**	**Android 2.3 (API Level 9)**	**Android 2.2 (API Level 8)**	**Android 1.5 (API Level 3)**
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a1	n/a
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes	Yes	Yes	Yes
TYPE_PRESSURE	Yes	Yes	n/a	n/a
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes	Yes	Yes	Yes

external apps. NODE+ [nod] and Sensorbug [sena] are wireless sensor platforms developed to be used with mobile devices. They provide documentation with the specification needed to operate its sensors but the number and type of built-in sensors is similar to the ones we already find in a smartphone.

2.4 Technology

In this section, the technologies used in this application both for the development of this native Android application and the protocols and standards that allow the smartphone communication with other components will be presented. At the end we also present the standards followed for the user interfaces.

2.4.1 Development technologies

Android is a mobile operating system based on the Linux kernel developed by Google, primarily for smartphones and tablet computers. This is currently the number one system in terms of shipped devices and is predicted that its share will continue to increase as shown in table 2.2 .

Table 2.2: Worldwide Device Shipments by Operating System (Thousands of Units) [sma]

Operating System	**2014**	**2015**	**2016**
Android	1,156,111	1,454,760	1,619,030
iOS/Mac OS	262,615	279,415	298,896
Windows	333,017	355,035	393,256
Others	626,358	380,545	261,155
Total	2,378,101	2,469,755	2,572,338

For our implementation we will use the **Android Software Development Kit** (SDK), it contains a set of tools and packages to work with all the features offered by this operative system. We'll use the **Android Studio** as developing IDE and **Java** as the programming language since that is the android standard. For the graph design we will use the Graphview[gra]. This is a graph plotting library with some implemented methods and structures that assist on the design of static and real-time graphs.

2.4.2 Messaging protocols

In our implementations design the smartphone will communicate with both the Selcomp device and the Cloud. For this purpose we studied some messaging protocols that may solve our problem. We will first describe each protocol and how it operates and than make a comparison between them.

- **DDS** [dds] or Data Distribution Service was first developed for the Aerospace and Defense community to address the requirements of mission-critical systems. This standard aims to enable scalable, real-time, dependable, high performance and interoperable data exchanges, and it is being increasingly used in a wide range of intelligent manufacturing applications [Fos].

- **AMQP** [amqa] or Advanced Message Queuing Protocol is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security [SS07]. Its goal is to enable a wide range of different applications and systems to be able to work together, regardless of their internal designs, standardizing enterprise messaging on industrial scale [amqb].

- **JMS** [jms] or Java Message Service is a Java Message Oriented API for sending loosely coupled, reliable, and asynchronous messages between two or more clients. It is one of the most widely used publish-and-subscribe messaging technologies and it supports both point-to point and publish-and-subscribe style routing [SS07]. In the point-to-point messaging system, messages are routed to an individual consumer which maintains a queue of "incoming" messages. This model supports publishing messages to a particular message topic. The main disadvantage of this implementation is that it is a Java API standard only.

- **MQTT**, or Message Queuing Telemetry Transport, is a publish-and-subscribe oriented messaging protocol for use on top of the TCP/IP protocol, developed by Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech) in 1999 for the monitoring of an oil pipeline through the desert [MQTb]. This is a lightweight protocol designed to operate in remote locations with constrained devices and low-bandwidth, high-latency or unreliable networks [MQTa]. It is ideal for machine to machine (M2M) communication and for mobile applications where bandwidth and battery consumption are very important aspects. In this model we have three

different entities, the MQTT broker and the publisher and subscriber clients. The publisher sends its data to the broker with a given topic, the broker will then push this data to all clients that subscribed to that specific topic.

- **REST**, Representational State Transfer or RESTful Web services are one way of providing interoperability devices over the Internet. This protocol is based on the request-response model where clients make requests and the server retrieves a respective response. The kind of operations available in this model include those predefined by the HTTP verbs GET, POST, PUT, DELETE. REST systems aim for fast performance, reliability, and scalability, by re-using components that can be managed and updated without affecting the whole system.

- **CoAP** [coa] or Constrained Application Protocol is used for simple devices in constrained networks, allowing them to connect to the Internet. CoAP is an application layer protocol that provides a request/response interaction model, supports built-in discovery of services and includes key concepts of the Web such as URIs. It runs over UDP (not TCP) with support for multicast, it has a very low overhead and it's simple to use making it usable for constrained environments.

In table 2.3 a comparison between all these protocols is presented, based on the following criteria: the abstraction; architecture; Quality of Service; Interoperability; Performance; Real-time support; Transport protocols; Support for mobile devices; Security. Based on these we are able to choose which protocols can better fit our dissertation requirements.

2.4.3 QR Code

QR Code or Quick Response Code, is a two-dimensional symbol invented in 1994 by Denso Wave Incorporated, one of major Toyota group companies, and approved as an ISO international standard in June 2000. Initially this 2D symbol was intended for use in automotive industry, but it has spread to other fields [Soo08]. Related to 1D barcode it presents a wider range of encoding, a larger capacity, confidentiality, anti-counterfeiting and error correction.

This 2D barcode can be scanned by a common smartphone with an embedded camera sensor and a software able to perform the decode and read the meta-data in it. The use of this code in mobile applications can have many different purposes: share a link, advertising, share personal contact, share Wi-Fi password, website login, payment code or check a product's information are just a few examples of this technology's application.

2.4.4 NFC

NFC or Near Field Communication is a technology that allow a wireless and secure communication between two devices at close range (approximately 5 centimeters), using short range radio waves. This technology is originally derived from RFID and it operates in a frequency range centered on 13.56 MHz with transmission rate of up to 424 kbit/s [Min11].

Table 2.3: Summary of Key Comparison Criteria [Fos]

	DDS	**MQTT**	**AMQP**	**JMS**	**REST/HTPP**	**CoAP**
Abstraction	Pub/Sub	Pub/Sub	Pub/Sub	Pub/Sub	Request/Reply	Request/Reply
Architecture style	Global Data Space	Brokered	P2P or Brokered	Brokered	P2P	P2P
QoS	22 Policies	3 Policies	3 Policies	3 Policies	Provided by transport e.g. TCP	Confirmable or nonconfirmable messages
Interoperability	Yes	Partial	Yes	No	Yes	Yes
Performance	10s of 1000s of messages per second.	100s to 1000+ messages per second per broker	100s to 1000+ messages per second per broker	100s to 1000+ messages per second per broker	100s of requests per second	100s of requests per second
Real-time	Yes	No	No	No	No	No
Transports	UDP by default but other transports such as TCP can also be used	TCP	TCP	Not specified but typically TCP	TCP	UDP
Dynamic discovery	Yes	No	No	No	No	Yes
Mobile devices	Yes	Yes	Yes	Dependent on JAVA capabilities of the OS	Yes	Via HTTP proxy
Security	Typically based on SSL or TLS with proprietary access control	Simple Username/ Password Authentication, SSL for data encryption	SASL authentication, TLS for data encryption	Typically based on SSL or TLS	Typically based on SSL or TLS	DTLS

This technology was developed by Phillips, Sony and Nokia at the first NFC forum. This forum is a non-profit organization with more than 150 members among companies and academics aiming on the development of standards to the use of NFC. As it can be observed in the figure 2.3 the number of NFC-enabled smartphones is continuously growing. Statista [staa] predicts that in 2018 there would be 1.9 billion phones with NFC support.

Traditionally, NFC has been used for applications like virtual wallets for commercial transactions, domotics, access control and ID cards, car keys or file and data transfer between two devices, but other usages for this technology are actively being developed.

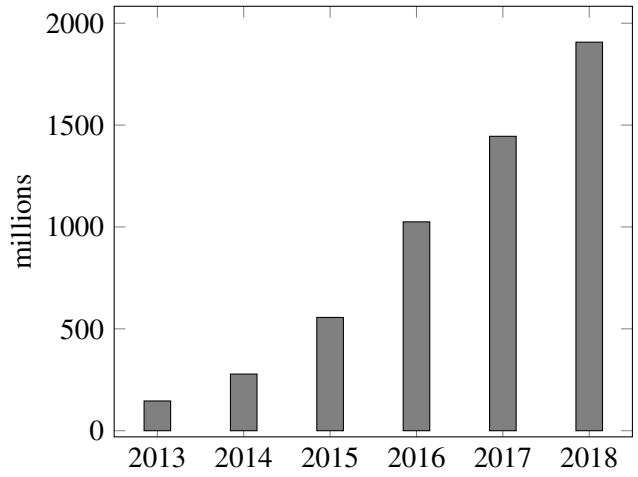

Figure 2.3: NFC-enabled phones worldwide from 2013 to 2018 [staa]

2.4.5 Google Material Design

Material design is a design language launched by Google in 2014 at the Google I/O conference. It "synthesizes the classic principles of good design with the innovation and possibility of technology and science" [MDe]. This concept opened the design standards used by Google, in their applications, to general developers.

2.5 Related Projects

Given its significance to the evolution of the worldwide industry, there is an high interest in Intelligent Manufacturing. In this regard, several companies are working on new products that can work with smartphone applications. These projects have similar functionalities, since they follow the same goals on assisting operators to better understand machines and production processes as well as interact with them. Some examples of these are listed below.

- **Tulip** [tul] presents some solutions for shop-floor apps, assisting operators in different steps of the production process. Helping operators on dynamic work instructions, quality control and help to visualize the hole state of the production system or for example the data related

to a sensor network attached to it. This company also offers the possibility to customize all its applications to match all its clients constraints.

- **[x]cube LABS** [xcu] is a company with several solutions for the IoT. In the manufacturing systems they have mobile apps to assist on checking inventory, production and supply chain overview and system surveillance, aiming for a full automated and connected factory.

- **BOSCH** [bos] also presents some Industry 4.0 solutions aiming on bring the shop floor data more accessible to operators. Through the use of mobile applications they try to improve the process quality and make processes more flexible. On released mobile application gives assistance at the maintenance process.

- **Prodsmart** [pro] is a portuguese company whose software allows the analysis of production lines in real-time. This projects makes use of the mobile devices to acquire information at the shop floor that is processed in real-time improving the whole system efficiency.

- **Altizon** [alt] has the Datonis cloud platform where all devices connect and stream its data to. This data is available to query, search and analytics anywhere. In the cloud the data is processed, analyzed and machine learning techniques are applied in order to improve the production system.

2.6 Summary

In this chapter we presented the state of the art related to the topics and technologies which are the basis of this thesis. Selsus and its precedent projects, developed solutions in order to solve some industrial challenges by using the Internet of Things paradigm, in which all devices are connected to the network and data can be easily visualized and processed, to optimize production processes or aid in the machine maintenance and diagnosis.

Smartphones are one of the most used mobile devices and are able to run complex applications. They have a growing number of built-in sensors and the ability to connect to some external ones. From our research we found that most of this external sensors have their own applications to work with and don't provide their APIs, this way we are not able to use these sensors. The solution we found that provides its specification interface has the same type of sensors from the smartphone so we discarded this option.

Some messaging protocols were reviewed in order to understand which would be the best fit for the smartphone communications with other devices in the Selsus environment. From those in section 2.4.2 we will use MQTT since it was developed to work in this IoT solutions, where machines and sensor networks are constantly generating large quantities of data. We will use REST to communicate with the cloud since it's the protocol already in use in Selsus implementation.

There are some related projects that are using the smartphone as an interface to access the system's information like the ones in section 2.5. Our application will not only be used as an

interface but also as a tool for machine diagnosis and calibration, using the smartphone sensors to collect new external data about the machines and the whole production system's state.

Chapter 3

Selsus Manufacturing Systems

The vision of Selsus [Sel], Health Monitoring and Life-Long Capability Management for SELf-SUStaining Manufacturing Systems, is to "create a new paradigm for highly effective, self-healing production resources and systems". It aims to develop new production systems where machines are aware of their state and history creating a diagnostic and prognostic environment.

The shop floor has a lot of different components with different sensing and data processing capabilities, its integration requires some standardization. Selsus developed a framework that allows communication between all this different devices in a way that they are aware of each other and the overall system status.

Two key areas explored by this project are the Wireless Sensor Networks (WSN) and the use of cloud computing in the manufacturing industry. Sensor captured data is very important on the machine behavior modeling and process optimization since it has the potential to provide extremely useful information that may be used to predict system failure, effectively schedule maintenance and optimize efficient energy use. Cloud Systems provide shared computer processing resources and data on demand. The combination between sensor networks and cloud computing led to the arrival of sensor clouds.

Within the sensor cloud the smart sensor nodes have the ability to collaborate with other smart sensor nodes, to store and process data, interpret events, communicate with different networks and be tolerant to failures [Fer15]. In this chapter an overview of the Selsus project is presented.

3.1 Selsus architecture

We start by describing this system's architecture and how different modules operate and communicate. In this design there are three main modules, the Sensor Network, that runs through machines acquiring different types of data, the Sensor Cloud, where all data is stored and processed, and the Selcomp which is connected to groups of sensors and establishes the communication between the sensors and the cloud. For a better understanding of this system, we present a diagram with Selsus architecture below.

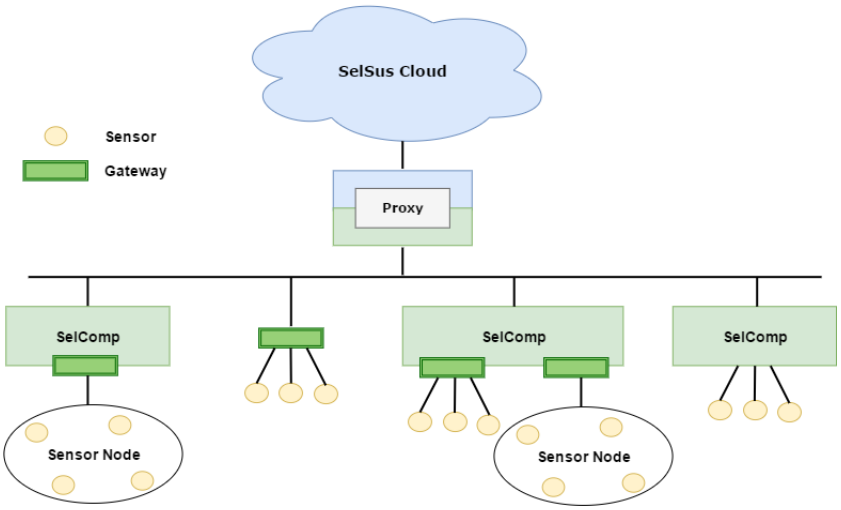

Figure 3.1: Selsus architecture

Sensors can have many functionalities and purposes. For that reason they have different architectures and distinct ways of acquiring and sharing its data. Selsus developed a framework where communication standards allow all these sensors to be connected to the system. In the figure 3.1 groups of sensors and Sensor Nodes have different ways to connect to the system, according to its capability to communicate with Selsus standards. Sensor Nodes are groups of sensors that are able to pass information through its network to a main location. When sensors are not able to communicate with Selsus, a gateway establishes this connection, making the translation of sensor data to the Selcomp.

In this design we have two levels of communication, one for the sensor/Selcomp module and another for the cloud. Both levels are coupled via a "proxy" which acts as a bridge for the interconnection of shop-floor and system level.

3.2 Selsus cloud

From all the sensors and actuators in the network large quantities of data are generated. This information will be stored in a cloud computing system named Selsus Cloud. This concept allows for all data to be accessed either synchronously or asynchronously over time and in any location with an internet connection.

Beside its storage capabilities and easy access to information, this sensor cloud design has a process unit where physical hardware and software can be intelligently perceived, and the collected information can be intelligently processed and used in the whole life-cycle of production [whi].

Some of the sensors in the network generate large amounts of data since the capture can be run at a high sample rate. For this reason, sensors are connected to Selsus Components. The Selcomp is a virtual representation of the physical equipment operating at the shop-floor level with built-in processing capabilities. This devices process the received data sending only some metrics to

the cloud. The general idea is to have all the manufacturing process machines represented by Selcomps in a virtual system, sending all its generated data to the Selsus Cloud.

3.3 Selcomp

The Selcomp concept goal is to create an innovative smart component, with enhanced reconfiguration, sensing, decision making and collaborative capabilities [whi]. This smart component is responsible for linking the sensor network and the Selsus cloud, as it is able to process the incoming sensor data and synchronize it with the cloud. To the Selsus system, Selcomps are perceived as machines since all data generated by the physical machines enter the system through this component.

These devices are able to collaborate with each other sharing important information of the overall state of the system. They can analyze a specific situation making decisions and changing their behaviour based on the feedback received from other Selcomps. This capacity to adapt through different scenarios is very important for the Selsus project goals.

This solution relies on several base technologies often used in IoT solutions. In figure 3.2, the Selcomp architecture is presented through its different modules.

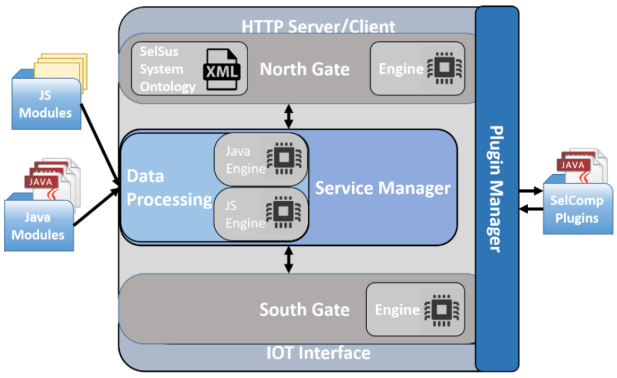

Figure 3.2: Sensor Selcomp architecture [whi]

- **North Gate:** This module is responsible for the Selcomp communication with other devices in the network allowing the Selcomp to be aware of the surrounding system. This is also the module that will connect to our application on order to send and receive sensor data.

- **Service Manager:** This module processes the data incoming from both the sensors and the other external components.It makes a bridge from the North and South Gates.

- **South Gate:** This is the sensor network module where sensors can connect either by wire or radio frequency. This module is able to receive new plugins so more devices can be integrated.

- **Plugin Manager:** All this different modules need well-defined interfaces to communicate. New plugins can be applied to extend capabilities.

3.3.0.1 Selcomp Recipe Adjustment

For the Selsus system the Selcomp is a representation of a physical machine. These devices are able to receive and process external machine data in order to alter or optimize its operational mode. This additional data allows the Selcomp processing unit to understand if its current behaviour should be changed or not.

This model turns the smartphone into a diagnosis tool since the smartphone embedded sensors are able to capture the external machine data that will processed by the Selcomp. The standards and document structure for this communication can be found in section 3.4.3.

3.4 Data structures

The communication between the different Selsus components requires some standardization. In this section we will describe the XML-based data structures used by our application to send data to the Selsus cloud and Selcomp.

This approach enables all the SelSus components to represent its dynamics in a common view to all the other components and the SelSus Cloud for easy interpretation and data generation. The specification for these data structures is presented in the appendix A, where a XML example is provided.

3.4.1 Selcomp Self-Description

The Selcomp Self-Description document describes the structure of the information to be sent and synchronized in the SelSus Cloud. Therefore, each Selcomp needs to create this document detailing the meta-information that will be generated during run-time so all the components that will receive information from this instance can know beforehand how to correctly interpret the information. An example of this file is presented in section A.1.1. This file has a large number of nodes, for this reason we will focus the most important for the context of our application. In figure 3.3 we have the node diagram to this file.

- **general:** This node has a generic structure composed by a set of data elements, where an ID and a value must be provided. Using this structure, any type of information can be defined, not constraining the Selcomps to have a predefined type of information like Serial Number, MAC Address and others.

- **deviceset:** Generally, an industrial machine and Wireless Sensor Network is composed by a set of components or sensors that generate information and are stored in a database for further analysis. The main purpose of the present node is to define the structure used to

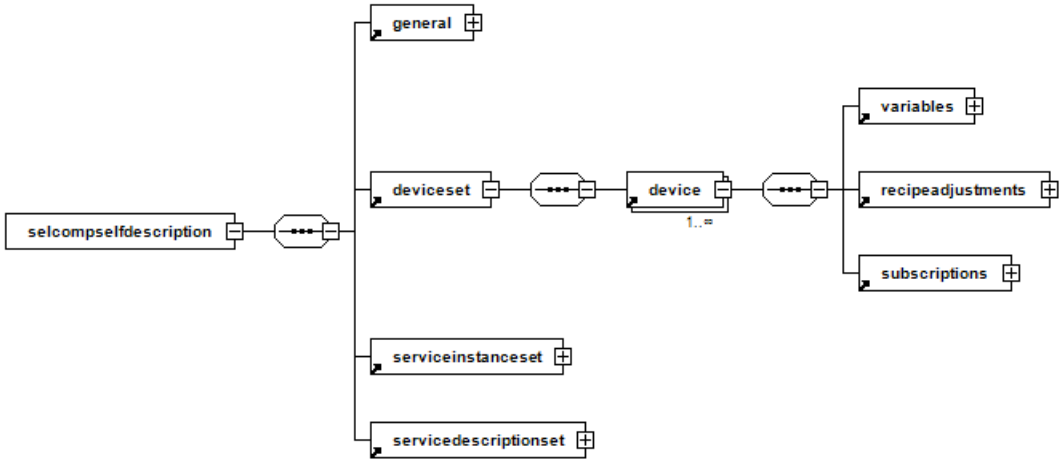

Figure 3.3: SSD structure diagram

represent the data of those components. Therefore, this node is composed by a set of device nodes, each one representing a machine component or integrated sensor in the system.

- **recipeadjustments:** In certain machines or Wireless Sensor Network there can exist pre-defined configurations that represent different kinds of shop-floor operations, from types of products being manufactured to execution modes like Limp Mode, Maintenance or normal execution. The present node is composed by a set of these operations (recipeAdjustment) that describe all the possibilities of component configuration.

3.4.2 Payload

The information which is synchronized with the SelSus Cloud or sent to other components needs to be totally aligned with the meta-information specified in the SSD so the correct interpretation of it can be made. Therefore, a structure that supports the defined meta-information was created, this XML structure is detailed as follows and a example of this file is presented in section A.1.2.

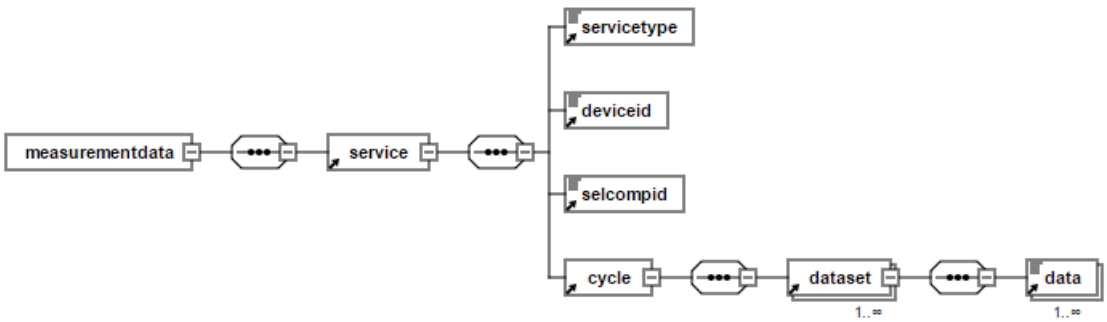

Figure 3.4: Payload structure diagram

- **Measurement Data:** This is the root node of the XML file composed by a set of service nodes, being these related with the type of information each device is generating: Observation, Maintenance, Diagnosis, Prognosis, Service and Self-Healing.

- **Service:** This is the node where the generated information of a certain type is kept. This means that if there are two devices generating Observation data, there will two service nodes with the serviceType attribute as “OBSERVATION”, and with the same deviceIDs defined in the SSD. The remaining information of the service node is composed by a set of cycle nodes, corresponding the cycle to one machine production cycle, described by a number attribute. This allows to analyze generated data from different Selcomps at the system level.

- **Dataset:** The dataset node is composed by data elements where the generated data is placed, according to the variables defined in the SSD. These data elements are a one-to-one correspondence of the variables defined in the device variables.

3.4.3 Recipe Adjustment

In certain machines or Wireless Sensor Network there can exist predefined configurations that represent different kinds of shop-floor operations, from types of products being manufactured to execution modes. Selcomps are able to receive this recipes adjusting its behaviour according to the data received. This files XML structure is detailed as follows.

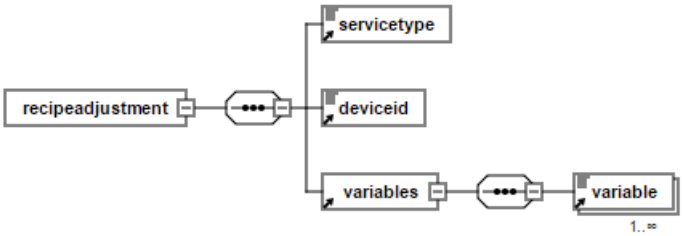

Figure 3.5: Recipe Adjustment structure diagram

- **Recipe Adjustment:** This node has three child nodes, the first two are the "serviceType" and the "deviceID" that should match the definition set in the SSD file, the last child is a set of variables that give information about how the component can be parameterized.

- **Variable:** This node is the description of a single variable used to describe the run-time generated data. It is composed by a name and a set of elements, being each described as an unique identifier (ID) and a value or array of values. This way, a certain variable can be described using a set of relevant descriptors that can provide information about

3.5 Back-end

The Sensor Cloud offers a set of services that enable its communication with the remaining Selsus devices. Through the use of HTTP requests the cloud is able to send and receive data. The Selcomp also provides a HTTP service to receive data. In these requests our application sends XML files using the SelSus communication protocol but also receives JSON files as answere to some particular requests. These services are specified in the following tables.

- **Get Selcomps:** Request a description of the Selcomps connected to the cloud.

Table 3.1: Get Selcomps request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Get Selcomps Cloud	GET	/systec_panel/service/getSelcompsCloud.php	-	JSON

  - **Response:** The response to this request is a JSON file with the cloud available sensors. An example of this file can be found in the section **??**.

- **Get Cloud data:** Request cloud data for a time interval with a given device ID.

Table 3.2: Get data request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Get Data Cloud	GET	/systec_panel/service/getDataCloud.php	deviceID, init_date, final_date	JSON

  - **deviceID:** Represents the device ID;
  - **init_date:** The start date for the capture requested;
  - **final_date:** The final date for the capture requested;
  - **Response:** The response to this request is a JSON file with the sensor data for the requested time interval. An example of this file can be found in the section **??**.

- **Post SSD file:** Post a Selcomp Self Description to the cloud, each new Selcomp must send its description to the Cloud.

Table 3.3: Selcomp Self Description POST request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Selcomp registration	POST	/systec_panel/service/registration	SSD	-

  - **SSD:** This parameter is a String with the XML SSD file. An example of this file can be found in section A.1.1.

- **Post Payload file:** Post a XML file with a sensor data capture to the Cloud.

Table 3.4: Payload POST request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Set Payload Cloud	POST	/systec_panel/service/payload	payload, comment	-

  - **payload:** This parameter is a String with the XML Payload file. An example of this file can be found in section A.1.2.
  - **comment:** This parameter is a String comment on the captured data.

- **Post media file:** Post a photo or video file to the Cloud.

Table 3.5: Recipe Adjustment POST request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Set Recipe Selcomp	POST	/WebServiceTest.asmx?WSDL/parseRecipe	recipe	-

  - **recipe:** This parameter is a String with the XML Recipe Adjustment file. An example of this file can be found in section A.1.3.

- **Post image file:** Post a photo from a specific machine to the Cloud .

Table 3.6: Image POST request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Post image file	POST	/systec_panel/service/image	deviceID, image, timestamp, comment	-

  - **deviceID:** This parameter is a String with the Selcomp ID;
  - **image:** This parameter is a file in the image/jpeg mime type;
  - **timestamp:** This parameter is a String with the capture timestamp.
  - **comment:** This parameter is a String with a comment on the picture.

- **Post video file:** Post a video from a specific machine to the Cloud .

Table 3.7: Video POST request

Endpoint	HTTP Method	URI	Request Parameters	Response Type
Post video file	POST	/systec_panel/service/video	deviceID, video, timestamp, comment	-

- **deviceID:** This parameter is a String with the Selcomp ID;
- **video:** This parameter is a file in the video/mpeg mime type;
- **timestamp:** This parameter is a String with the capture. timestamp.
- **comment:** This parameter is a String with a comment on the video.

3.6 Summary

Selsus makes use of the latest IoT technologies to build self-sustainable production systems. Through the use of sensor networks and cloud computing systems this project aims to help the European industry by building solutions for smart factories.

This project's architecture, back-end services and data structures allow the integration of the smartphone as a new smart component, the mobile Selcomp, the development of this concept is the objective of this dissertation. The use of smartphones can not only bring the system interface closer to the operator, but also aid in the system diagnosis and calibration.

Chapter 4

Industrial manufacturing app

The result of this dissertation is an Android application to be used in a shop-floor environment. This app will grant easy access to machine sensor data and to the general vision of the whole production system. This data can be retrieved from the history or real-time generated by the machine's embedded sensors.

The application is able to capture data from a set of built-in sensors which can be used by the Cloud for further processing or by the Selcomp to make some readjustment or to calibrate a specific sensor. This features allow machine operators to better understand the state of the system and perform easier and faster diagnostics.

To the Selsus environment a smartphone running our application is perceived as a new Selcomp, for this reason and in this project's context we named this devices as Mobile Selcomps. Like a Selcomp, before exchanging data with the Sensor Cloud, the smartphone must register itself in it, share its own information and a description of all devices connected to it.

The down-time of a machine is partially caused by the time it takes for a specialized technician to arrive and make the diagnosis and maintenance. But in some cases the machine can be experiencing a simple problem that could be solved from a distance after analyzing a photo or a video. The camera and microphone built in the smartphone allow operators to record images and videos, as well as sending them to the Cloud, where they are associated with a specific machine. This allows anyone with Internet access to be able to check these photos and videos and solve the problem at a distance.

In this section we will describe the Mobile Selcomp, starting with the underlying requirements of this sort of applications. We will then present the system's architecture and all the components within the Selsus environment which directly interacts with the mobile app. At the end we present some external libraries used in this implementation.

4.1 System requirements

For a better understanding of the developed application some functional and non functional requirements were identified.

4.1.1 Functional requirements

Functional requirements are the objectives that the different system components must fulfill. We will divide these requirements through the different modules that are part of the application.

- **Selcomp recognition requirements:** these requirements are related with the multiple ways to the application connect with a specific Selcomp device.

Table 4.1: Selcomp recognition requirements

ID	Name	Description	Priority
FR01.01	Get Selcomps	The application should present a list of the Selcomps available in the Cloud.	Essential
FR01.02	Selcomps search	The application should allow a text search over the Selcomps list.	Desirable
FR01.03	NFC recognition	The application should allow to connect to a Selcomp through a NFC tag recognition.	Important
FR01.04	QR Code recognition	The application should allow to connect to a Selcomp through a QR code recognition.	Important

- **Diagnosis tool requirements:** presents the requirements related with the Diagnosis tool both from the photo/video capture to capturing sensor data and sending it to the Cloud.

Table 4.2: Diagnosis tool requirements

ID	Name	Description	Priority
FR02.01	Record a photo/video	The application should allow to record a photo/video and send it to the Cloud.	Essential
FR02.02	Comment a photo/video	The application should allow to add a comment to a captured image or video.	Essential
FR02.03	Sensor info	The application should allow the access to information from smartphone embedded sensors.	Desirable
FR02.04	Sensor real-time data	The application should allow the access to near real-time data from smartphone embedded sensors.	Essential
FR02.05	Record sensor data	The application should allow to record data from the smartphone embedded sensors and send it to the Cloud.	Essential

- **Calibration tool requirements:** Here are presented the requirements for the Recipe Adjustment process.

Table 4.3: Calibration tool requirements

ID	Name	Description	Priority
FR03.01	Available sensors	The application should display the smartphone embedded sensors registered as recipe adjustments.	Essential
FR03.02	Sensor info	The application should allow access to information from the smartphone embedded sensors registered as recipe adjustments .	Desirable
FR03.03	Sensor real-time data	The application should display the selected smartphone embedded sensor's data in real-time.	Essential
FR03.04	Record sensor data	The application should allow to record data from the smartphone embedded sensors within the recipes list and send it to the Selcomp.	Essential

- **Selcomp Overview requirements:** presents the real-time data from a specific Selcomp connected sensor.

Table 4.4: Selcomp Overview requirements

ID	Name	Description	Priority
FR04.01	Display Selcomp sensors	The application should display the Selcomp available sensors	Essential
FR04.02	Display sensor data	The application should display a graph with the real-time data being generated by the selected sensor.	Essential

- **Cloud Overview requirements:** presents the machine data from a specific device, for a given period of time.

Table 4.5: Cloud Overview requirements

ID	Name	Description	Priority
FR05.01	Get Selcomp sensors	The application should display the Selcomp available sensors	Essential
FR05.02	Get sensor data	The application should request and display the historical data from the Selcomp selected sensor.	Essential

4.1.2 Non-functional requirements

Non-functional requirements concern the use of the application following some criteria, for example, availability, performance and versatility. The following requirements are the ones we found more significant:

- **NFC.01** The application should run in any smartphone running the Android OS version 1.5 (Level 3 API) or higher.
- **NFC.02** The smartphone sensors must be supported by Android SDK (check table 2.1).
- **NFC.03** The smartphone should have NFC support to use NFC recognition feature.
- **NFC.04** Data requests should not be perceivable to the user in terms of the time took to visualize a response.
- **NFC.05** Services should support multiple connections from different different application instances.

4.1.3 Actors

To better understand our problem requirements we need to know the different users that will use our application. Our actors are the machine operator and the maintenance engineer, both have the same permissions and can use all the application features. The maintenance engineer can use our app to access machine historical data during a diagnosis process or to capture some data with the smartphone sensors and send it to the Cloud for further process and analysis. The machine operator uses all the solution features.

Table 4.6: System actors

Name	**Permission**
Machine Operator (MO)	This user can access all application modules
Maintenance Engineer (ME)	This user can access all application modules

4.1.4 User stories

In the table 4.7 the user stories considered for the proposed solution are presented, as well as their effort estimation, using the Fibonacci scale, and their priority. Since both actors are able to use the same features we will use "GU" to represent a general user.

4.1.5 Use cases

The use case model allows to co-relate existing actors with the actions these can perform. We divided the use cases in different groups according to the application module they belong to.

Table 4.7: User stories.

ID	**Name**	**Description**	Priority	Effort
US01	Selcomps list	As a GU, I want a list from all Cloud active Selcomps.	Essential	3
US02	Selcomp search	As a GU, I want to text search for a available Selcomp.	Desirable	3
US03	QR code Selcomp select	As a MO, I want to select a Selcomp from a QR code.	Important	4
US04	NFC Selcomp select	As a GU, I want to select a Selcomp by scanning an NFC tag.	Important	5
US05	Photo/Video capture	As a ME, I want to take a photo or a video and send it to the Cloud.	Essential	5
US06	Photo/Video comment	As a ME, I want to add a comment to a recorded photo or video.	Important	2
US07	Smartphone sensor list	As a GU, I want to see a list with all sensors available to be used.	Essential	3
US08	Sensor info	As a GU, I want to check the smartphone embedded sensors info.	Desirable	3
US09	Sensor real-time data	As a GU, I want to visualize the data being generated by smartphone built-in sensors.	Essential	8
US10	Sensor capture data	As a ME, I want to use the smartphone embedded sensors to capture data and send it to the Cloud.	Important	5
US11	Sensor capture comment	As a MO, I want to attach a comment to the capture.	Desirable	3
US12	Selcomp sensors	As a GU, I want a list of the Selcomp's connected sensors.	Essential	3
US13	Selcomp real-time data	As a MO, I want to visualize data being generated by the Selcomp connected sensors.	Essential	8
US14	Cloud sensor data	As a ME, I want to retrieve sensor data from the cloud for a specified time interval.	Essential	5

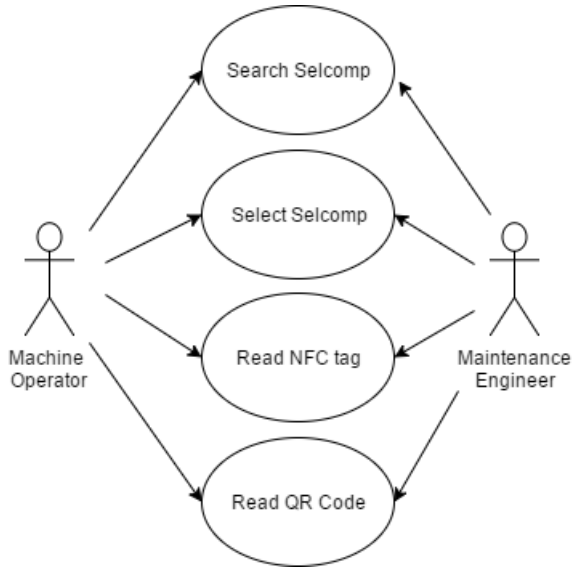

Figure 4.1: Selcomp recognition use cases diagram.

4.1.5.1 Selcomp recognition

Table 4.9: UC02 - Selcomp select

ID	UC02
Name	Select Selcomp
Description	Allow the user to select a Selcomp to connect, from those available in the Cloud.
Actor	MO, ME
Preconditions	The Cloud service should be available
Postconditions	The application has all Selcomp info.
Procedure	Click on the desired Selcomp (C.3)

Table 4.10: UC03 - NFC tag read

ID	UC03
Name	Read NFC tag
Description	Allow the user to identify and select a Selcomp by reading an NFC tag.
Actor	MO, ME
Preconditions	Smartphone must support NFC
Postconditions	The Selcomp is selected
Procedure	1. Click the NFC floating button (C.3) 2. Pass the device close to the NFC tag (C.5)

Table 4.8: UC01 - Selcomp search

ID	UC01
Name	Search Selcomp
Description	Allow the user to search for a string filtering the Selcomp list.
Actor	MO, ME
Preconditions	Selcomp's list must be loaded
Postconditions	Filtered Selcomps list.
Procedure	1. Click the search icon (C.4) 2. Type the string

Table 4.11: UC04 - QR code read

ID	UC04
Name	Read QR code
Description	Allow the user to identify and select a Selcomp by reading a QR code.
Actor	MO, ME
Preconditions	Smartphone must have a camera
Postconditions	The Selcomp is selected
Procedure	1. Click the QR code floating button (C.3) 2. Scan the QR code

4.1.5.2 Diagnosis tool

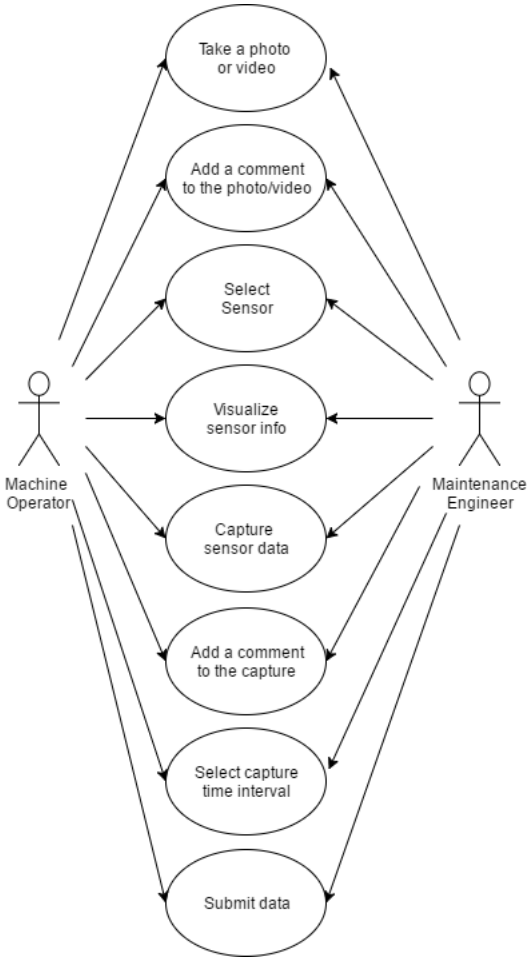

Figure 4.2: Diagnosis tool use cases diagram.

Table 4.12: UC05 - Photo/Video capture

ID	UC05
Name	Take a photo or video
Description	Allow the user to capture a photo or a video and send it to the cloud.
Actor	MO, ME
Preconditions	Smartphone must have a camera sensor.
Postconditions	The media is stored in the Cloud.
Procedure	1. Click on the photo or video button (C.10) 2. Press the capture button 3. Click the send button (C.21)

Table 4.13: UC06 - Comment media

ID	UC06
Name	Add a comment
Description	Allow the user to add a comment to a photo or video.
Actor	MO, ME
Preconditions	The photo/video must already be taken.
Postconditions	The comment is added to the media file.
Procedure	1. Click on Add comment (C.21) 2. Type the content of the comment 3. Click on the save button

Table 4.14: UC07 - Select sensor

ID	UC07
Name	Select sensor
Description	Allow the user to select a sensor from the smartphone sensors list.
Actor	MO, ME
Preconditions	Required sensor must be available.
Postconditions	The sensor is selected.
Procedure	Click on the desired sensor. (C.10)

Table 4.15: UC08 - Sensor info

ID	UC08
Name	Visualize sensor info
Description	Allow the user to check sensor info and the real-time generated data.
Actor	MO, ME
Preconditions	The sensor must be selected.
Postconditions	The info is displayed.
Procedure	Scroll through the different tabs (C.11, C.12, C.13)

Table 4.16: UC09 - Capture data

ID	UC09
Name	Capture sensor data
Description	Allow the user to capture data from the smartphone embedded sensors.
Actor	MO, ME
Preconditions	Sensors must be supported by the Android SDK.
Postconditions	The data is saved.
Procedure	1. Select the recording tab (C.13) 2. Click start capture button (C.13) 3. Click on the stop button (C.14)

Table 4.17: UC10 - Comment graph

ID	UC10
Name	Capture comment
Description	Allow the user to add a comment to the captured sensor data.
Actor	MO, ME
Preconditions	Data must be captured already.
Postconditions	The comment is added to the captured data.
Procedure	1. Click on Add comment (C.15) 2. Type the content of the comment 3. Click on the save button

Table 4.18: UC11 - Select time interval

ID	UC11
Name	Select time interval
Description	Allow the user to set a time interval on the captured data to send.
Actor	MO, ME
Preconditions	The data must be captured already.
Postconditions	The time interval is set.
Procedure	Slide the time bar to the desired interval (C.15)

Table 4.19: UC12 - Submit data

ID	UC12
Name	Submit data
Description	Allow the user to send the captured data to the Cloud for further processing and analysis.
Actor	MO, ME
Preconditions	Cloud service must be available.
Postconditions	The data is stored in the Cloud.
Procedure	Click the send button (C.15)

4.1.5.3 Calibration tool

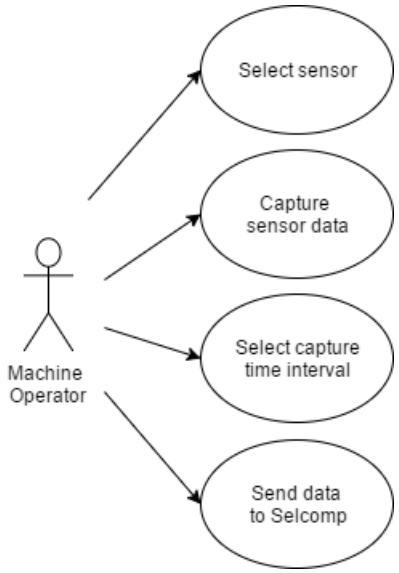

Figure 4.3: Calibration tool use cases diagram.

Table 4.20: UC13 - Select calibration sensors

ID	UC13
Name	Select sensor
Description	Allow the user to select a smartphone sensor within those present on the SSD file.
Actor	MO
Preconditions	The sensors must be registered as a recipe adjustment.
Postconditions	The sensor is now selected.
Procedure	Click on the desired sensor (C.16)

Table 4.21: UC14 - Capture calibration data

ID	UC14
Name	Capture sensor data
Description	Allow the user to capture data from the selected sensor.
Actor	MO
Preconditions	The sensor must be selected.
Postconditions	Data captured for that period.
Procedure	1. Click start capture button (C.13) 2. Click on the stop button (C.14)

Table 4.22: UC15 - Select data time interval

ID	UC15
Name	Set time interval
Description	Allow the user to set a time interval on the captured data to send.
Actor	MO
Preconditions	The data must be captured already.
Postconditions	The time interval is set.
Procedure	Slide the range seek bar to the desired time interval (C.15)

Table 4.23: UC16 - Submit calibration data

ID	UC16
Name	Send data to Selcomp
Description	Allow the user to send the captured to the Selcomp.
Actor	MO
Preconditions	Selcomp services should be available.
Postconditions	Start of the recipe adjustment process.
Procedure	Click in the send button (C.15)

4.1.5.4 Cloud Overview

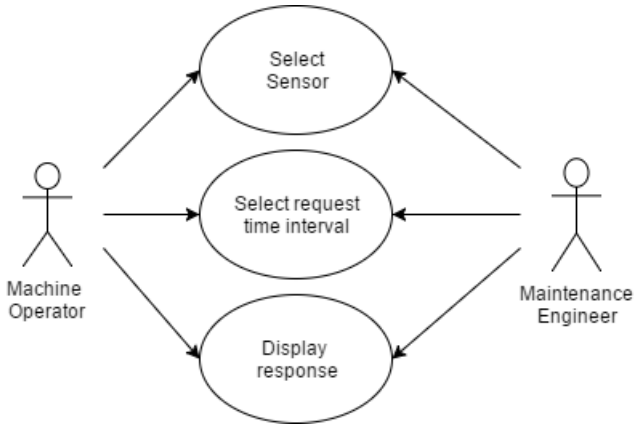

Figure 4.4: Cloud overview use cases diagram.

Table 4.24: UC17 - Select sensor

ID	UC17
Name	Select sensor
Description	Allow the user to select the sensor to overview.
Actor	MO, ME
Preconditions	Cloud services must be on.
Postconditions	The sensor is selected.
Procedure	Select the desired sensor (C.16)

Table 4.25: UC18 - Select request time interval

ID	UC18
Name	Select request time interval
Description	Allow the user to set a time interval in the requested sensor data.
Actor	MO, ME
Preconditions	The data must be captured already.
Postconditions	The time interval is set.
Procedure	Slide the time bar to the desired interval (C.18)

Table 4.26: UC19 - Display graph

ID	UC19
Name	Display response graph
Description	Allow the user to visualize a graph with the requested sensor data.
Actor	MO, ME
Preconditions	The data must be captured already.
Postconditions	The time interval is set.
Procedure	Analyze the graph (C.19)

4.1.5.5 Selcomp Overview

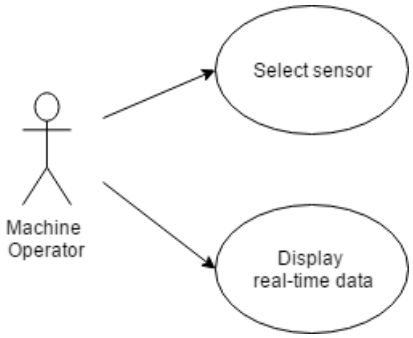

Figure 4.5: Selcomp overview use cases diagram.

Table 4.27: UC20 - Select Selcomp sensor

ID	UC20
Name	Select sensor
Description	Allow the user to select a sensor form the Selcomp's sensors list.
Actor	MO
Preconditions	Sensor services must be on.
Postconditions	The sensor is selected.
Procedure	Click in the desired sensor (C.16)

Table 4.28: UC21 - Select time interval

ID	UC21
Name	Display real-time data
Description	Allow the user to see the Selcomp's sensor data in a real-time graph.
Actor	MO
Preconditions	MQTT client must be set.
Postconditions	The app is receiving the sensor data.
Procedure	Visualize the data (C.20)

4.2 System architecture

As seen in the previous chapter the developed application was built on top the project SelSus existing model. In its architecture this project has two main modules: the Selcomp and the Sensor Cloud. These devices allow the shop-floor machines virtualization, with a huge amount of data being generated by all its connected sensors. Our application will interact with both of these devices in order to fulfill all its requirements. To the Selsus environment the smartphone is perceived as a new Selcomp since it uses the existing services to interact with the other system's components, this way the current Selcomp and Sensor Cloud's architecture doesn't need any update to allow the smartphone integration.

In the figure 4.6 there is a representation of our solution's architecture, where the smartphone has an established connection with both the SelComp module and the Sensor Cloud. These are the interfaces that provide all the data to the application and both will be described in the sections below. The interface between the Selcomp and the Cloud is a mean to store all the sensor network data. On the opposite direction the Cloud uses this interface to manage and send recipe adjustments to the Selcomp. The interfaces with the smartphone will be explained in the next sections.

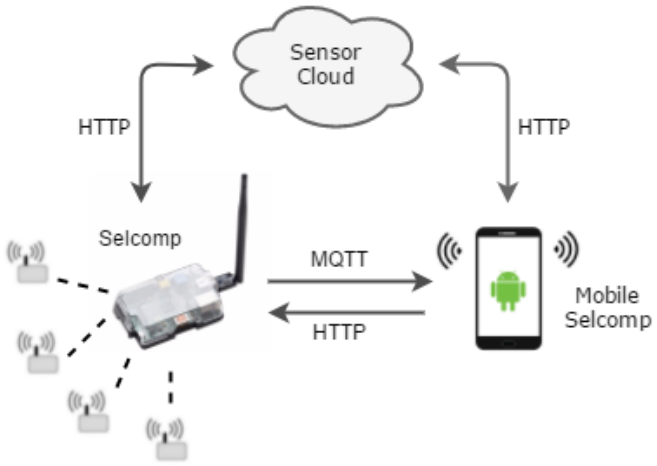

Figure 4.6: Solution architecture

4.3 Selcomp selection

In a system with a large number of machines we have a large number of Selcomps. In order to easily connect these machines we thought of different ways our application can recognize a Selcomp. When the app starts an HTTP request (3.1) is sent from the smartphone to the Cloud requesting the information on all Selcomps connected to it. In this request's response (A.2.1), the Cloud retrieves all the information related to its connected Selcomps as well as all the sensors connected to them. From this response we also get all recipe adjustments that each Selcomp is prepared to run.

After receiving this information a list with all the available Selcomps is presented in the app 4.7. The search in this list can be made by three different ways: text search for the device name, scan an NFC tag or read a QR Code. In the case the users knows the Selcomp name, search for it could be the fastest way, in the case he doesn't or he wants to, one of the other two options should be used. From the QR code and NFC actions a device ID is returned and a match is made with the device IDs within the Selcomps list.

With the Selcomp selected the app presents a menu (4.8) with this different application modules. Those are described in the following sections.

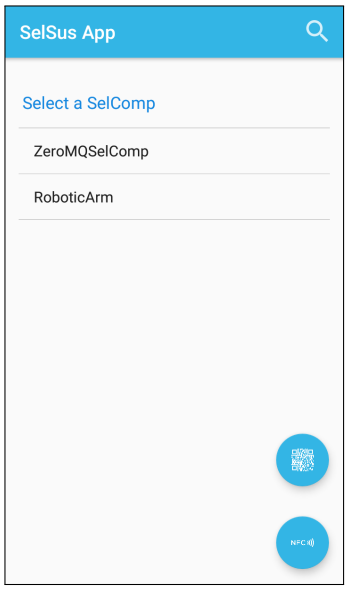

Figure 4.7: Selcomp select activity

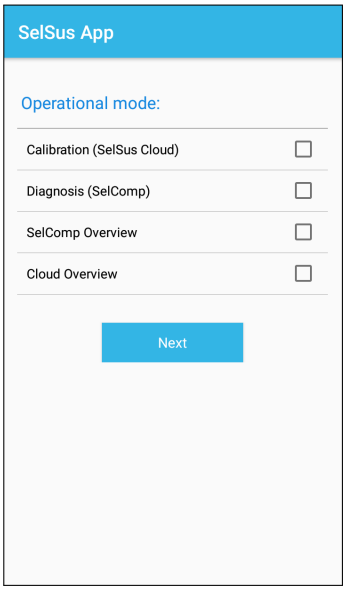

Figure 4.8: Mode select activity

4.4 Sensor Cloud interface

In the Sensor Cloud all data related to the production system is stored. This interface allows our application to access this data and present it to the user. In the opposite direction the smartphone is also able to send the data generated from its embedded sensors to the Cloud. These are two of our application modules that we will describe in the subsections below. For this connection we use the HTTP/REST protocol. This, besides being already implemented in the cloud, is simple to use and has a request/response abstraction that matches our problem constraints on data exchange.

4.4.1 Cloud overview module

This module allows users to request data stored in the Sensor Cloud. To obtain this data the application makes an HTTP GET request (A.2.2) to the Cloud services sends as parameters the device ID and the time interval. After entering this module the user must choose the sensor he wants to overview (figure 4.9). With the sensor already selected, the user is able to set the time interval for the requested sensor data. There are different predefined time intervals for the user to select or the day as we can see in figure 4.10. When the "Other" option is checked the user may set a specific time interval setting the "From" and "To" dates with a time picker (figure 4.11).

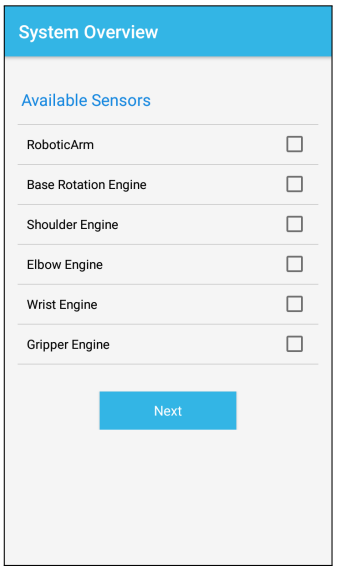

Figure 4.9: Selcomp sensor select activity

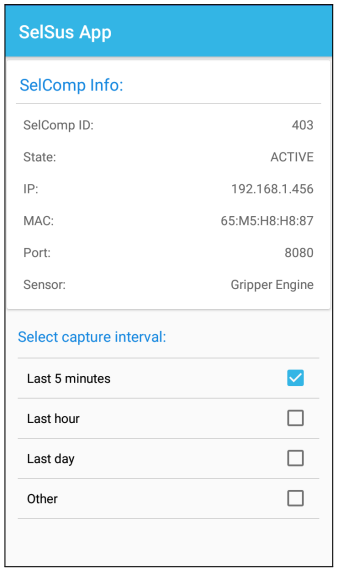

Figure 4.10: Request time interval activity

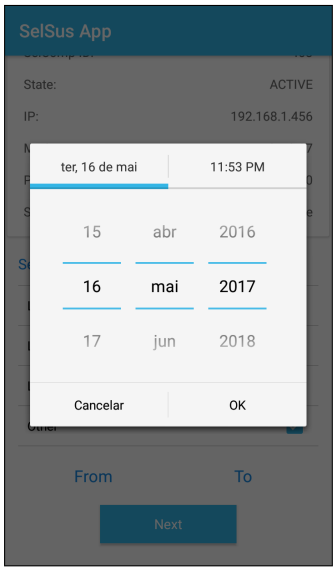

Figure 4.11: Date and time picker

With all the parameters set the request is sent to the Cloud. The response is a JSON Object containing a set of timestamps as keys and the captured data as value (A.2.2). This data is processed and transformed into a graph. All the keys and value turn into Data Points, a Java structure from the Graphview plotting library [gra] that can store two doubles as X and Y values. Sets of this Data Points turn into graph series. In the figure 4.12 we can see an example of one of these graphs.

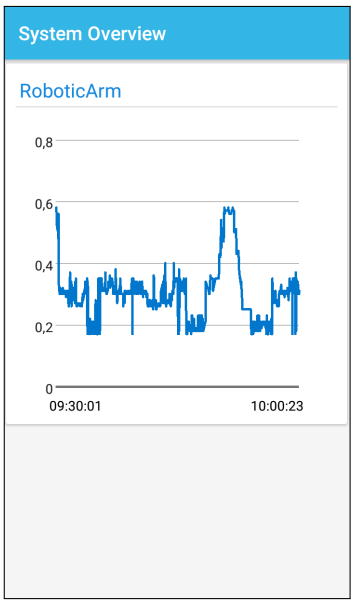

Figure 4.12: Cloud graph result

4.4.2 Diagnosis module

In this operational mode we try to aid the user to understand what's causing a machine failure. This is achieved by gathering additional information on the machine which can be further accessed by other people or processed in the Cloud. When selecting this mode the user has a list of the smartphone built-in sensors 4.13 that he can select in order to overview or capture its real-time generated data. In the same activity it is possible to use the camera sensor to capture a picture or a video from a faulty machine and send it to the Cloud.

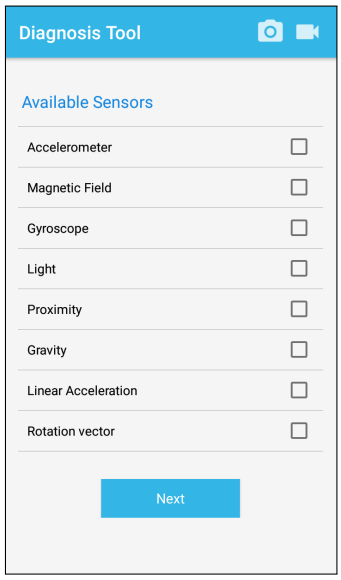

Figure 4.13: Smartphone sensors list

After a sensor is selected its data is presented in a tab menu. The first tab presents the sensor real time data (figure 4.14), the second has information about the sensor hardware (figure 4.15) and the third presents a graph with the real-time generated sensor data (figure 4.16). To obtain all this information we use the Android SDK Sensor methods.

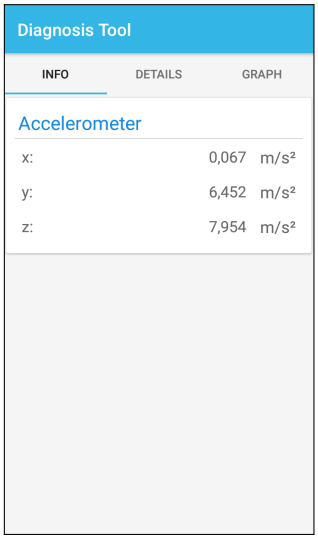

Figure 4.14: Sensor real time info

Figure 4.15: Sensor hardware details

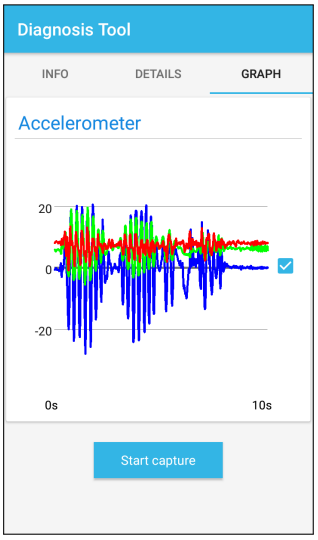

Figure 4.16: Sensor real-time graph

At the end of the activity 4.15 the "Start capture" button allows the user to start recording the sensor data being displayed in the graph. After this process, the capture time interval can be adjusted and a comment could be added to it like we can observe in figure 4.17. Before sending this data an HTTP POST request (3.3) must be sent to the Cloud to register itself and its sensors, this information is sent as a request parameter and it is a string containing the Selcomp Self-Description file (A.1.1). With all set, data can be sent to the Cloud through another HTTP POST, as parameters is sent the payload file (3.4) with the acquired data.

In this module the user is able to capture some media files and associate it with a specific machine. The buttons on top of this mode's main activity (4.13) launch the Android camera activity, allowing the user to capture photos and videos. After the capture operation the user is redirected to our app to handle its results. The photo or video taken is displayed and a comment can be added to it (4.18). With all prepared, a HTTP POST request is sent to the Cloud with two parameters: the device ID and the media file.

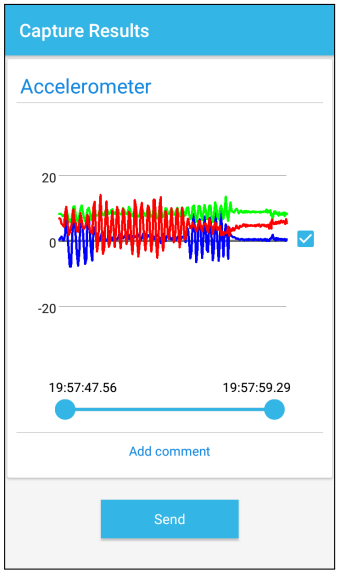

Figure 4.17: Graph results activity

Figure 4.18: Photo/video capture results

4.5 Selcomp interface

This interface is responsible for two of our application modules. The first is the Selcomp Overview where the user is able to check the real-time generated data from a specific sensor. To achieve this we use the MQTT protocol, a publish/subscriber based messaging protocol that allow our app to subscribe to any Selcomp publishing. The second module is the Calibration mode where the smartphone captures data from its embedded sensors in order to perform a recipe adjustment operation. The exchange of communication from the smartphone to the Selcomp is made by HTTP requests.

4.5.1 Selcomp overview module

In this module the smartphone is able to present the real-time data being generated by the Selcomp's sensors. The incoming data is received using the MQTT protocol, which is often used in IoT projects related with sensor networks since it is very light and it's optimized to work over low fidelity TCP/IP networks.

The data generated from the Selcomp connected sensors is being published in a MQTT Broker using its device ID as topic. In this module a list of the Selcomp's available sensors is presented to the user (figure 4.9). After selecting one of these sensors, the app subscribes to that sensor's topic and starts receiving its captures. This data is presented to the user in a real-time graph displaying the last 10 seconds of sensor activity (figure 4.19).

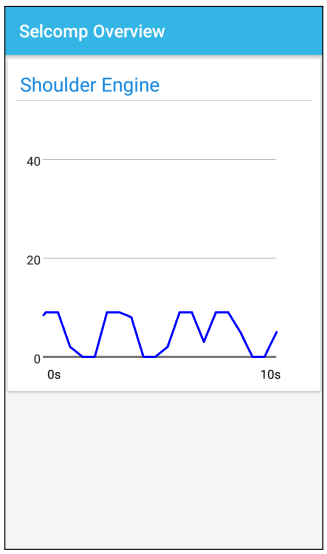

Figure 4.19: Real-time Selcomp sensor graph

4.5.2 Calibration module

This module's operation aims at the correction of the machine functioning through some adjustments in its sensors and actuators. This is achieved using the SelSus recipe adjustment, where the Selcomp has the ability to receive data from an external source, and use it to change its operational mode in order to resolve a machine problem or to prevent future machine failures.

After selecting this module a list of sensors is presented to the user. These sensors are the smartphone built-in sensors reported as available to recipe adjustment in that Selcomp's Self-Description file (A.1.1). As we can see in the figure 4.20, from all smartphone built-in sensors, only the accelerometer sensor can be used in a recipe adjustment operation.

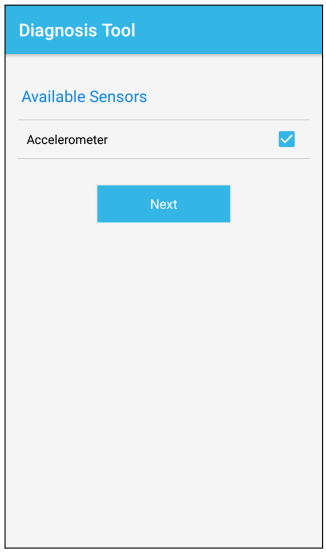

Figure 4.20: Selcomp sensors available for recipe adjustment

With the sensor selected the user is able to record data, that it will further send to the Selcomp. The capture process is similar to the Diagnosis mode where the user starts/stops the data acquisition and is able to set the capture's time interval or even add a comment, before sending it to the Selcomp 4.17. The acquired information is sent through an HTTP POST request with the recipe adjustment file (3.4.3) as parameter.

4.6 External libraries

In this section we present some external libraries used in the development of our application.

- **GraphView:** this is an open source graph plotting library which we use to display the sensors data.
- **ZXingScannerView:** this library provides a barcode scanner that uses the smartphone camera sensor to process a QR code.
- **Volley:** It is an HTTP library by Google, that makes networking for Android apps easier.
- **XMLBuilder:** It is a Java library which allows the easy creation of XML documents.
- **Paho Android Service:** It is an MQTT client library written in Java for Android applications.

4.7 Summary

We started this chapter by describing our solution requirements, starting with the functional and non-functional requirements description, followed by definition of the actors and all operations associated with them. After this we presented our solution's architecture which contains the Mobile Selcomp integration with SelSus devices and the interfaces between the smartphone and this components.

Chapter 5

Integration and validation of the industrial app in the sensor cloud

In order to validate our solution, some scenarios have been created allowing us to verify the fulfillment of this application requirements. In the Institute for Systems and Robotics' lab we prepared a setup where we can observe the behaviour of the developed app in a real case scenario. In this setup we have a robotic arm generating operational data, a Sensor Cloud, a Selcomp and smartphone, all connected in the same network.

In this chapter, we present our validation scenario's architecture, with a description of the devices in it, as well as a set of tests where we can verify if our functional requirements are satisfied. We divided these tests by our application modules.

5.1 Validation architecture

In this section we will present our system physical and logical architectures. In our design we have three different nodes. The first, is a desktop running an instance of the Sensor Cloud and an instance of a Selcomp device, this offers the services needed to test the different application modules. The second, is the robotic arm able to generate data from its movements and send it to other devices. The third is the smartphone that will connect to this network and exchange data with other devices. A diagram with this architecture is presented in figure 5.1.

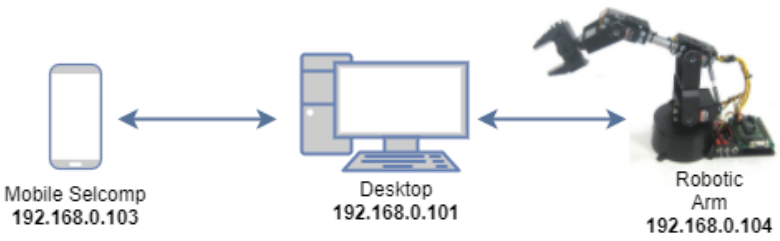

Figure 5.1: System physical architecture

Logically, our system is divided in three distinct layers: the Collaborative Environment, the DigitalTwins and the Cloud Infrastructure. The first layer, are the physical devices generating data to the system, in our scenario the robotic arm and the smartphone. The second one, is a virtualization of all this physical devices allowing them to share its acquired data or receive data from the Cloud. Also in this layer there is the connection between the Mobile Selcomp and the Machine Selcomp supported by the MQTT messaging protocol. In the Cloud Infrastructure layer, we have three different modules: the Cloud services which allows other devices to communicate with the Cloud and access its data, a storage unit where data is physically stored and a processing module that handles the workflow and the integration of new sensors and modules. A diagram with this architecture is presented in figure 5.2.

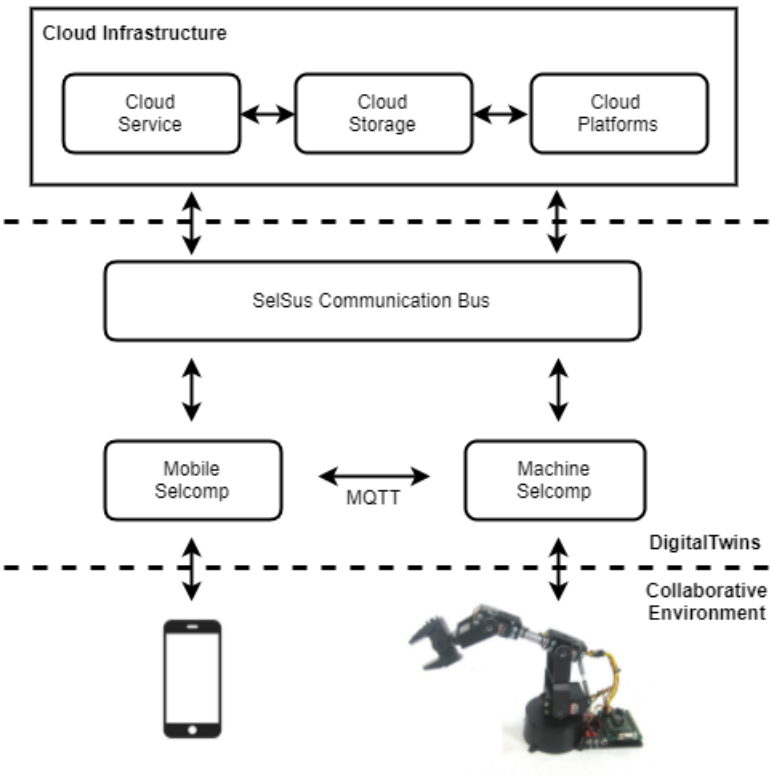

Figure 5.2: System logical architecture

5.2 Validation scenarios

5.2.1 Scenario 1

In this first scenario we will use our application to connect to a specific Selcomp. Before this operation can be performed the application must request a JSON file with the Sensor Cloud's available devices. The figure 5.3 presents a screen capture from the desktop's command line running the Cloud services, where we can check the smartphone incoming request. An example of the Cloud's

response file can be found in the appendix B.1.1 and it contains a list of the Cloud available Selcomps, the sensors connected to each one of them and the recipe adjustments available. A list of this sensor is presented in the table 5.1.

```
192.168.0.101 - "GET /systec_panel/service/getSelcompsCloud.php HTTP/1.1" 200 -
```

Figure 5.3: Cloud incoming get Selcomps request

Table 5.1: Selcomps registered in the Cloud

Selcomp	**selcompID**	**IP**	**MAC**	**Port**
RoboticArm	426	192.168.0.104	02:00:4C:4F:4F:50	57681
GTP_Demo_SelComp	425	192.168.1.500	65:M5:H8:H8:87:00	8080

For this test we setup two NFC tags and two printed QR codes with the ID of each Selcomp available in the Cloud. With this we will verify if our application is able to match one of these IDs with those received from the Cloud response (table 5.1). The QR codes used for this test can be found in the appendix section B.1.2. With an ID match, this activities redirect to the operational mode menu activity. For the 2 NFC tags and both QR codes tested, the recognition process was successful and the app was able to select the corresponding Selcomp.

Another way the user can select a Selcomp is by searching for its name at the search bar. To demonstrate this, in figure 5.4 we can verify that the string passed as input is filtering the Selcomp's list that initially had 2 different Selcomps. With this test the functional requirement defined for the Selcomp recognition, presented in table 4.1, were satisfied.

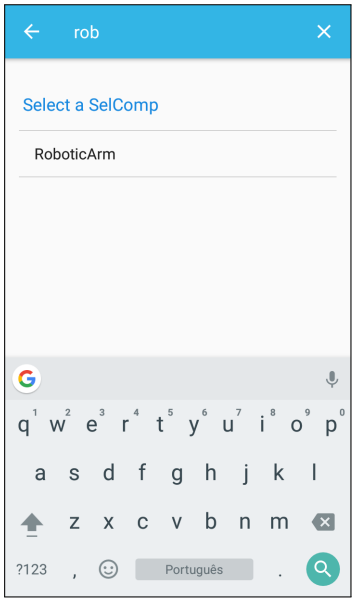

Figure 5.4: Selcomps text search

5.2.2 Scenario 2.a

In this scenario we will perform two similar actions: capture a photo and a video, which will later be sent to the Sensor Cloud. In the shop-floor, this operation could be used to capture machine failures or malfunctions for further review, here we just took a photo of our robotic arm as an example, as we can see in figure 5.5. Before sending this files to the Cloud, a comment is added. In figure 5.6 we can see a screen capture from the desktop running the Cloud services, when the POST requests are received.

Figure 5.5: Media results activity

```
192.168.0.101 - "POST /systec_panel/service/image HTTP/1.1" 200 -
192.168.0.101 - "POST /systec_panel/service/video HTTP/1.1" 200 -
```

Figure 5.6: Cloud incoming media POST requests

5.2.3 Scenario 2.b

The Diagnosis mode also allows users to store data that's being acquired by the smartphone embedded sensors. In this test we will capture some of this data, in this case, generated by the accelerometer sensor.

After a sensor is selected, we slide to the real-time graph tab and start recording the sensor data for about 10 seconds. The result of this capture is presented in figure 5.7. The capture's time interval is then set to the first 5 seconds, a comment is added and the data is sent to the Cloud. An

example of the recipe XML file generated is presented in section A.1.2 . A screen capture, from the Cloud's POST request received, is presented in figure 5.8.

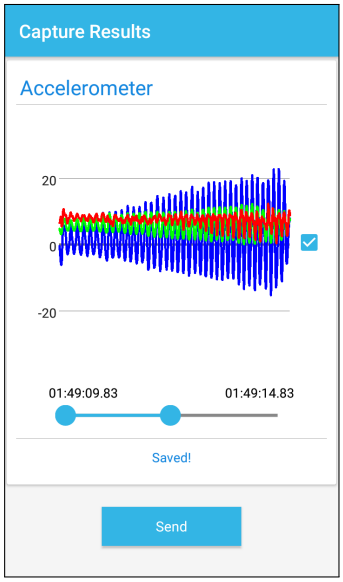

Figure 5.7: Diagnosis capture results

```
192.168.0.101 - "POST /systec_panel/service/paylod HTTP/1.1" 200 -
```

Figure 5.8: Cloud incoming payload POST requests

5.2.4 Scenario 3

In this scenario we will use our application to calibrate the movement of a robotic arm when performing a simple task. The smartphone is placed on top of the robotic arm's base rotation engine, so it can follow its movements and capture the data being generated by its built-in accelerometer.

Each Selcomp has a list with the different recipe adjustments that can be performed. In the Selcomps description file received from the cloud (B.1.1) we can verify that the robotic arm is able to receive accelerometer data as a recipe as we can see in the following snippet.

```
1    "115595": {
2          "name": "ProcessAdjustment",
3          "variables": {
4            "Accelerometer1": "Integer",
5            "Accelerometer2": "Integer",
6            "Accelerometer3": "Integer",
7            "Timestamp": "Integer"
8          }
9        }
```

The robotic arm was programmed to transport one screw between two defined locations, in a an increasing velocity loop. This increment is the simulation of a machine failure or malfunction which, in our scenario, could be the deterioration of the base rotation engine. When the robotic arm is working correctly, its movements velocity should increase in the first operations but after a while it should stay uniform. In order to understand what is a normal behaviour, we previously provided the Selcomp with an accelerometer capture from a period where the machine was working properly and it is presented in figure 5.9, this capture will serve as comparison to future calibrations.

With the faulty operation running in the robotic arm (constantly increasing velocity), we start capturing data. The result of this capture is presented in figure 5.10. The Selcomp is able to compare both graphs and reconfigure its processes in order to correct this difference. In the graph, a robotic arm move generates a local maximum and its frequency is compared in both captures allowing the Selcomp to understand the number of movements for a time interval. In this case, the frequency registered while in a malfunction period is higher than the standard behaviour. This means that the machine is working to fast.

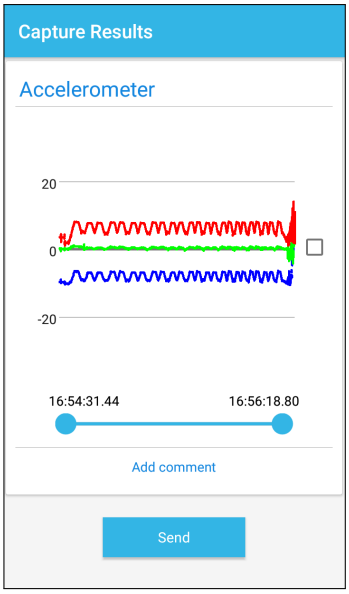

Figure 5.9: Normal functioning capture

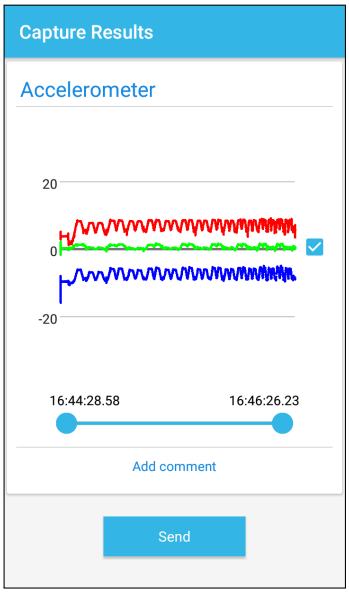

Figure 5.10: Malfunction capture

In order to correct its movements, the Selcomp slows down the base rotation engine. In figures 5.11 and 5.12 we present captures from the base rotation engine velocity (mm/s) before and after the recipe adjustment process. As expected, after this process, the base rotation engine's velocity increases until it reaches a maximum calculated value of 52 mm/s.

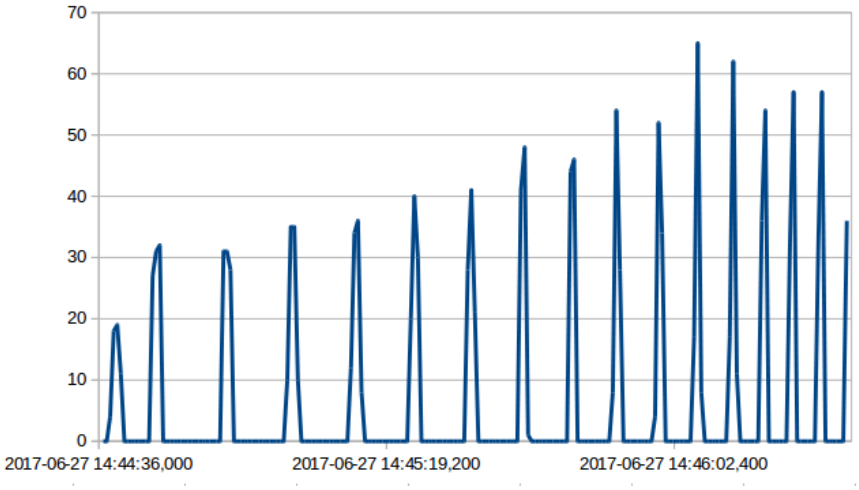

Figure 5.11: Base rotation engine before the recipe adjustment

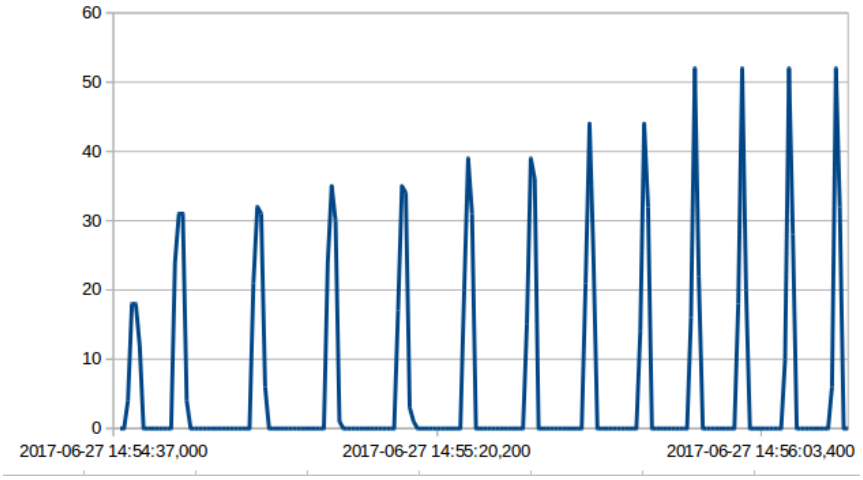

Figure 5.12: Base rotation engine after the recipe adjustment

5.2.5 Scenario 4

In this scenario we will use our application to retrieve some historical data from the Sensor Cloud. To do this we select a sensor, in this case the Base Rotation Engine, and a time interval to the requested data. With all set we click the "Send" button. The parameters set for this request are presented in table 5.2.

Table 5.2: Cloud data request parameters

Parameter:	**Value:**
deviceID	7830
init_date	2017-06-23 18:23:00
final_date	2017-06-23 18:24:59

The application sends this request and receives a JSON file with the sensor data as response, an example of this file can be found in section A.2.2. In figure 5.14 we have a print screen from the Cloud service receiving the GET request. In figure 5.13 a screen capture from this process resulting graphic is presented.

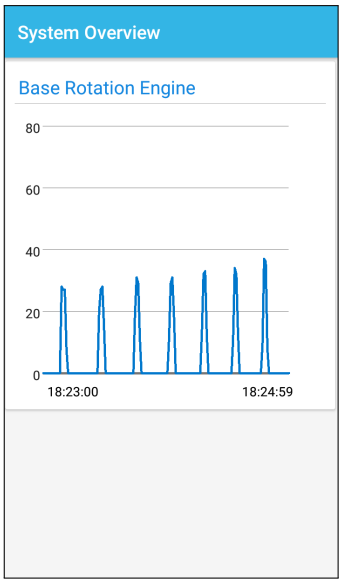

Figure 5.13: Sensor Cloud data

```
192.168.0.101 - "GET /systec_panel/service/getDataCloud.php HTTP/1.1" 200 -
```

Figure 5.14: Cloud get data request

5.2.6 Scenario 5

In this last scenario we will check the real time data generated by a Selcomp connected sensor. In run-time the Selcomp is publishing its generated data to a MQTT broker, making it available for any MQTT client subscribing to its topic. In the application, we select a sensor and, therefor, a MQTT client is launched subscribing to that sensor's topic, in this case, the sensor ID. These IDs can be found in the Selcomp's description file (B.1.1) which the following snippet belongs to.

```
"selcompID": "426",
     "sensors": {
       "Base Rotation Engine": "7830",
       "Elbow Engine": "7832",
       "Gripper Engine": "7834",
       "RoboticArm": "7835",
       "RoboticArmStop": "7836",
       "Shoulder Engine": "7831",
```

```
9         "Wrist Engine": "7833"
10      }
```

The MQTT client connection parameters are presented in the table 5.3. After the connection is established, a stream of data starts incoming and can be displayed in a real-time graph with a 10 seconds window. In figure 5.15 a screen capture is taken from this graph's activity.

Table 5.3: MQTT Client connection parameters

Sensor	Shoulder engine
IP	192.168.0.104
Port	57681
Topic	7831

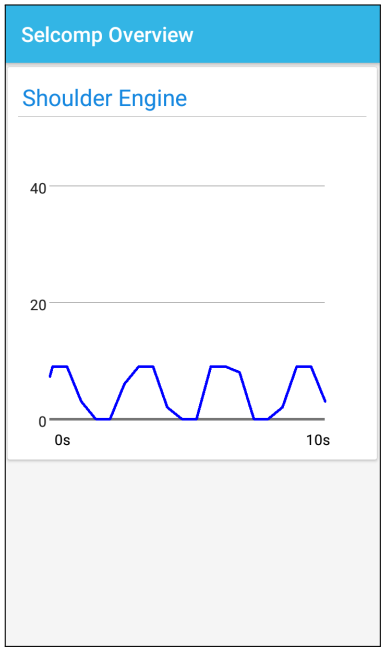

Figure 5.15: Shoulder engine real-time data

5.3 Summary

Our objective with this case test scenario was to verify the fulfillment of our application functional requirements. After running this tests, we are now able to conclude that our app meets all the predefined requirements, since all these tasks where successfully concluded. In the table 5.4 we can observe the functional requirements tested in each scenario.

Our application is able to connect to the Selsus environment exchanging data with its different devices from the Selcomp recognition, where information is acquired from the QR code and NFC tags, to the communication with the Sensor Cloud and the Selcomps within this system.

Table 5.4: Functional requirements fulfillment

Scenario 1	FR01.01, FR01.02, FR01.03, FR01.04, FR01.05
Scenario 2.a	FR02.01, FR02.02
Scenario 2.b	FR02.03, FR02.04, FR02.05
Scenario 3	FR03.01, FR03.02, FR03.03, FR03.04
Scenario 4	FR05.01, FR05.02
Scenario 5	FR04.01, FR04.02

Chapter 6

Conclusion and future work

In this chapter we present the conclusions for the developed work during this dissertation. In the beginning we will present the problem encountered, as well as the proposed solution. After this we approach the contribution given to the Industrial Internet of Things field and to SelSus. At last we present some future work.

6.1 Conclusion

One of the major problems in an industrial shop-floor is the machine downtime. This could be reduced by using better diagnosis and maintenance processes. In the SelSus project, some solutions have been developed regarding this issue. The SelSus Selcomp is the virtualization of a physical machine able to collect all the data being generated by all the machine sensors and actuators. This device is connected to a network that allows communication with other Selcomps and other components within this system. The Sensor Cloud is another SelSus component that provides a cloud computing service to the SelSus system where Selcomps can connect and exchange information among themselves. The Cloud has three main modules: the Cloud Service which provides the input and output APIs, the Cloud storage where all the data is recorded and the Cloud platform where the processing is done.

The objective of this dissertation was to create an Android mobile application on top of SelSus, providing machine operators and maintenance engineers with a set of tools to aid in its diagnosis and calibration processes, and serve as a mobile interface to all data being generated within this system. In order to exchange data with other devices, our application must follow the standards defined for the SelSus Selcomp and Sensor Cloud communication. To the Cloud, the smartphone is perceived as a new Selcomp able to receive or generate data from a set of embedded sensors.

Our solution is divided in different modules which serve different functionalities: The Diagnosis module allows users to capture data from the smartphone embedded sensors and to send it to the Cloud. This data can be further processed and used in reconfiguration processes. Also in this module the user is able to capture a photo or a video and send it to the Cloud associating it with a Selcomp. This can be used to record machine failures to further review or share it with other users

that may assist in solving the problem reported; The Calibration module allows machine operators to use our application to automatically reconfigure a machine operations from the data generated from the smartphone built-in sensors; The Cloud overview module allows for easy access to the Cloud historical data; The Selcomp overview module allows users two observe the real-time data being generated by the machine sensors.

In order to validate our solution we tested our app in a real case scenario. In it, we have a Cloud instance running, a Selcomp and a robotic arm, that will interact with our app throughout the different tests. Through these tests we were able to verify the fulfillment of all the functional requirements previously set for our application. The SelSus project aims to develop solutions for advanced production systems. With this validation we demonstrated that our application can be integrated in a shop-floor environment to be used as a diagnosis and calibration tool, assisting system users in their daily operations. This, should not only be seen as a test to the fulfillment of our application's functional requirements but also as the validation of the industrial mobile application concept and the smartphone usage within these systems. This work's contributions are presented in the following section.

6.2 Contribution

The contribution of this dissertation aims at different targets. In a shop-floor environment, the Cloud and Selcomp overview modules bring system interfaces closer to machine operators. A true understanding of the machine state is very important for the prevention of machine failures. The Calibration module aids machine operators to easily correct some machine malfunctioning through our app. The Diagnosis mode allows for more information to be added to the Cloud. This can be accessed by users trying to solve some problem with the machine or by the Cloud's processing unit. To the SelSus research team, this application can be of assistance when working with this system's different devices by displaying all the data being generated and stored within. To the scientific community, we hope to encourage more people to start exploring the smartphone sensing capabilities and developing new solutions within the IIoT context.

6.3 Future Work

The integration of smartphones in the industrial shop-floor is fairly recent and we are witnessing the development of the first mobile applications for this environment. Below we present some of the topics that could be explored in the future within our application context.

- In the application, display an album with all media files from a Selcomp. This way, users could search this album for specific machine problems.
- External sensors can be added to the application in order to provide a wider range of services.

- An usability study could be performed in order to understand the real impact of the application in a real shop-floor.
- Some security and robustness standards could be applied since these aspects deviate from the main focus of this dissertation.

Appendix A

Selsus files and data structures

A.1 Data Structures

A.1.1 Selcomp Self-Description

```
1
2  <?xml version="1.0"?>
3  <selcompselfdescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://selsus.eu/ssd">
4    <general>
5      <data id="SELCOMP_NAME">RoboticArm</data>
6      <data id="DEVICE_TYPE">SELCOMP</data>
7      <data id="SELCOMP_ID">9999</data>
8      <data id="SELCOMP_IP">192.168.10.128</data>
9      <data id="SELCOMP_PORT">57681</data>
10     <data id="SELCOMP_MAC">65:M5:H8:H8:87</data>
11   </general>
12   <deviceset>
13     <device id="115590" type="OBSERVATION">
14       <variables>
15         <variable name="Base Rotation Engine">
16           <elements>
17             <element id="Type">Voltage</element>
18             <element id="MeasurementType">V</element>
19             <element id="DataType">java.lang.Integer</element>
20             <element id="max">2500</element>
21             <element id="min">500</element>
22           </elements>
23         </variable>
24       </variables>
25       <recipeadjustments>
26         <recipeadjustment name="Base Rotation Engine Aquisition Rate">
27           <variables>
28             <variable name="SamplingRate">
```

```
            <elements>
              <element id="DataType">java.lang.Integer</element>
              <element id="units">milliseconds</element>
              <element id="min">0</element>
              <element id="max">999999</element>
            </elements>
          </variable>
        </variables>
      </recipeadjustment>
    </recipeadjustments>
    <subscriptions />
  </device>
  <device id="115591" type="OBSERVATION">
    <variables>
      <variable name="Shoulder Engine">
        <elements>
          <element id="Type">Voltage</element>
          <element id="MeasurementType">V</element>
          <element id="DataType">java.lang.Integer</element>
          <element id="max">2500</element>
          <element id="min">500</element>
        </elements>
      </variable>
    </variables>
    <recipeadjustments>
      <recipeadjustment name="Shoulder Engine Aquisition Rate">
        <variables>
          <variable name="SamplingRate">
            <elements>
              <element id="DataType">java.lang.Integer</element>
              <element id="units">milliseconds</element>
              <element id="min">0</element>
              <element id="max">999999</element>
            </elements>
          </variable>
        </variables>
      </recipeadjustment>
    </recipeadjustments>
    <subscriptions />
  </device>
  <device id="115592" type="OBSERVATION">
    <variables>
      <variable name="Elbow Engine">
        <elements>
          <element id="Type">Voltage</element>
          <element id="MeasurementType">V</element>
          <element id="DataType">java.lang.Integer</element>
          <element id="max">2500</element>
          <element id="min">500</element>
```

```
          </elements>
        </variable>
      </variables>
      <recipeadjustments>
        <recipeadjustment name="Elbow Engine Aquisition Rate">
          <variables>
            <variable name="SamplingRate">
              <elements>
                <element id="DataType">java.lang.Integer</element>
                <element id="units">milliseconds</element>
                <element id="min">0</element>
                <element id="max">999999</element>
              </elements>
            </variable>
          </variables>
        </recipeadjustment>
      </recipeadjustments>
      <subscriptions />
    </device>
    <device id="115593" type="OBSERVATION">
      <variables>
        <variable name="Wrist Engine">
          <elements>
            <element id="Type">Voltage</element>
            <element id="MeasurementType">V</element>
            <element id="DataType">java.lang.Integer</element>
            <element id="max">2500</element>
            <element id="min">500</element>
          </elements>
        </variable>
      </variables>
      <recipeadjustments>
        <recipeadjustment name="Wrist Engine Aquisition Rate">
          <variables>
            <variable name="SamplingRate">
              <elements>
                <element id="DataType">java.lang.Integer</element>
                <element id="units">milliseconds</element>
                <element id="min">0</element>
                <element id="max">999999</element>
              </elements>
            </variable>
          </variables>
        </recipeadjustment>
      </recipeadjustments>
      <subscriptions />
    </device>
    <device id="115594" type="OBSERVATION">
      <variables>
```

```
        <variable name="Gripper Engine">
          <elements>
            <element id="Type">Voltage</element>
            <element id="MeasurementType">V</element>
            <element id="DataType">java.lang.Integer</element>
            <element id="max">2500</element>
            <element id="min">500</element>
          </elements>
        </variable>
      </variables>
      <recipeadjustments>
        <recipeadjustment name="Wrist Engine Aquisition Rate">
          <variables>
            <variable name="SamplingRate">
              <elements>
                <element id="DataType">java.lang.Integer</element>
                <element id="units">milliseconds</element>
                <element id="min">0</element>
                <element id="max">999999</element>
              </elements>
            </variable>
          </variables>
        </recipeadjustment>
      </recipeadjustments>
      <subscriptions />
    </device>
    <device id="115595" type="OBSERVATION">
      <variables>
        <variable name="RoboticArm">
          <elements>
            <element id="Type">Operation</element>
            <element id="MeasurementType">boolean</element>
            <element id="DataType">java.lang.Integer</element>
            <element id="max">1</element>
            <element id="min">0</element>
          </elements>
        </variable>
      </variables>
      <recipeadjustments>
        <recipeadjustment name="ProcessAdjustment">
          <variables>
            <variable name="Time">
              <elements>
                <element id="DataType">java.lang.Integer</element>
                <element id="units">seconds/seconds</element>
                <element id="min">500</element>
                <element id="max">2500</element>
              </elements>
            </variable>
```

```
176             <variable name="Box">
177               <elements>
178                 <element id="DataType">java.lang.Integer</element>
179                 <element id="units">seconds/seconds</element>
180                 <element id="min">1</element>
181                 <element id="max">3</element>
182               </elements>
183             </variable>
184           </variables>
185         </recipeadjustment>
186 
187       </recipeadjustments>
188       <subscriptions />
189     </device>
190   </deviceset>
191   <tancesetserviceins>
192     <serviceinstance id="s0" type="OBSERVATION" dId="cloudService">
193       <subscriptions>
194         <id>115590</id>
195       </subscriptions>
196     </serviceinstance>
197     <serviceinstance id="s1" type="OBSERVATION" dId="cloudService">
198       <subscriptions>
199         <id>115591</id>
200       </subscriptions>
201     </serviceinstance>
202     <serviceinstance id="s2" type="OBSERVATION" dId="cloudService">
203       <subscriptions>
204         <id>115592</id>
205       </subscriptions>
206     </serviceinstance>
207     <serviceinstance id="s3" type="OBSERVATION" dId="cloudService">
208       <subscriptions>
209         <id>115593</id>
210       </subscriptions>
211     </serviceinstance>
212   </serviceinstanceset>
213   <servicedescriptionset>
214     <servicedescriptionset dId="cloudService" name="Cloud" version="1.0.0" type="
          ObservationType" namespace="non">
215     </servicedescriptionset>
216   </servicedescriptionset>
217 </selcompselfdescription>
```

A.1.2 Payload

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<measurementdata xmlns="http://selsus.eu/sensorData">
    <service>
        <servicetype>ObservationService</servicetype>
        <deviceid>115590</deviceid>
    <cycle number="1">
      <dataset>
          <data id="measurement">{-0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:25.315177Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{-0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:26.316397Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:27.315177Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,0.5349609375}</data>
          <data id="timestamp">2016-07-19T10:03:28.316397Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,0.6549609375}</data>
          <data id="timestamp">2016-07-19T10:03:29.316397Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,0.7849609375}</data>
          <data id="timestamp">2016-07-19T10:03:30.316397Z</data>
      </dataset>
    </cycle>
    <cycle number="2">
      <dataset>
          <data id="measurement">{-0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:25.315177Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{-0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:26.316397Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,-0.0849609375}</data>
          <data id="timestamp">2016-07-19T10:03:27.315177Z</data>
      </dataset>
      <dataset>
          <data id="measurement">{0.0849609375,0.0849609375,0.5349609375}</data>
```

```
            <data id="timestamp">2016-07-19T10:03:28.316397Z</data>
        </dataset>
        <dataset>
            <data id="measurement">{0.0849609375,0.0849609375,0.6549609375}</data>
            <data id="timestamp">2016-07-19T10:03:29.316397Z</data>
        </dataset>
        <dataset>
            <data id="measurement">{0.0849609375,0.0849609375,0.7849609375}</data>
            <data id="timestamp">2016-07-19T10:03:30.316397Z</data>
        </dataset>
    </cycle>
    </service>
</measurementdata>
```

A.1.3 Recipe Adjustment

```
<?xml version="1.0" encoding="UTF-8"?>
<recipeadjustment name="ProcessAdjustment">
  <servicetype>OBSERVATION</servicetype>
  <deviceid>13452</deviceid>
  <variables>
    <variable name="Accelerometer1">
      {2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7, 2.235170057929281E-7,
          2.235170057929281E-7, 2.235170057929281E-7}
    </variable>
    <variable name="Accelerometer2">
      {9.776309967041016, 9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
```

```
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016, 9.776309967041016, 9.776309967041016,
          9.776309967041016}
    </variable>
    <variable name="Accelerometer3">
      {0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228, 0.8123480081558228,
          0.8123480081558228, 0.8123480081558228}
    </variable>
    <variable name="timestamp">{234276.47812, 234342.09212, 234407.45712,
        234472.62112, 234537.22112, 234603.47212, 234668.25112, 234732.85212,
        234798.04612, 234863.38212, 234928.34812, 234993.58512, 235059.25512,
        235123.93712, 235189.72112, 235255.14912, 235320.03212, 235385.43012,
        235450.39412, 235515.78112, 235581.77912, 235647.51812, 235713.07112,
        235779.20112, 235843.82612, 235909.23812, 235973.76212, 236039.41212,
        236104.98012, 236171.10112, 236237.06912, 236302.10012, 236366.76112,
        236433.05712, 236498.32012, 236562.70112, 236628.47112, 236693.48912}
    </variable>
  </variables>
</recipeadjustment>
```


A.2 Back-end

A.2.1 Get Selcomps response


```

{
  "selcomps": {
    "ZeroMQSelComp": {
      "selcompID": "401",
      "state": "ACTIVE",
      "ip": "192.168.10.189",
      "mac": "70-54-D2-AB-8E-32",
      "port": "8080",
      "sensors": {
```

```
        "Visible Light2": "847711111610150118105115105981081O1",
        "Invisible Light2": "8477111116101501051101181051151059810810l",
        "Visible Light5": "847711111610153118105115105981081O1",
        "Invisible Light5": "8477111116101531051101181051151059810810l",
        "Visible Light0": "847711111610148118105115105981081O1",
        "Invisible Light0": "8477111116101481051101181051151059810810l",
        "Invisible Light1": "8477111116101491051101181051151059810810l",
        "Visible Light1": "847711111610149118105115105981081O1"
      },
      "recipes": []
    },
    "RoboticArm": {
      "selcompID": "403",
      "state": "ACTIVE",
      "ip": "192.168.1.456",
      "mac": "65:M5:H8:H8:87",
      "port": "8080",
      "sensors": {
        "RoboticArm": "115595",
        "Base Rotation Engine": "115590",
        "Shoulder Engine": "115591",
        "Elbow Engine": "115592",
        "Wrist Engine": "115593",
        "Gripper Engine": "115594"
      },
      "recipes": {
        "115590": {
          "name": "Base Rotation Engine Aquisition Rate",
          "variables": {
            "Accelerometer1": "Integer",
            "Accelerometer2": "Integer",
            "Accelerometer3": "Integer",
            "timestamp": "timestamp"
          }
        },
        "115591": {
          "name": "Shoulder Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115592": {
          "name": "Elbow Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115593": {
          "name": "Wrist Engine Aquisition Rate",
```

```
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115594": {
          "name": "Wrist Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115595": {
          "name": "ProcessAdjustment",
          "variables": {
            "Speed": "Integer"
          }
        }
      }
    }
  }
}
```

A.2.2 Get data response

```
{
  "2017-06-16 09:30:01": "[0.58]",
  "2017-06-16 09:30:03": "[0.52]",
  "2017-06-16 09:30:04": "[0.51]",
  "2017-06-16 09:30:05": "[0.51]",
  "2017-06-16 09:30:06": "[0.51]",
  "2017-06-16 09:30:07": "[0.48]",
  "2017-06-16 09:30:08": "[0.56]",
  "2017-06-16 09:30:09": "[0.34]",
  "2017-06-16 09:30:10": "[0.31]",
  "2017-06-16 09:30:11": "[0.31]",
  "2017-06-16 09:30:12": "[0.3]",
  "2017-06-16 09:30:13": "[0.32]",
  "2017-06-16 09:30:14": "[0.3]",
  "2017-06-16 09:30:15": "[0.31]",
  "2017-06-16 09:30:16": "[0.3]",
  "2017-06-16 09:30:17": "[0.3]",
  "2017-06-16 09:30:18": "[0.31]",
  "2017-06-16 09:30:19": "[0.31]",
  "2017-06-16 09:30:20": "[0.31]",
  "2017-06-16 09:30:21": "[0.31]",
  "2017-06-16 09:30:22": "[0.32]",
  "2017-06-16 09:30:23": "[0.3]",
```

```
  "2017-06-16 09:30:24": "[0.31]",
  "2017-06-16 09:30:25": "[0.31]",
  "2017-06-16 09:30:26": "[0.31]",
  "2017-06-16 09:30:27": "[0.3]",
  "2017-06-16 09:30:28": "[0.3]",
  "2017-06-16 09:30:29": "[0.31]",
  "2017-06-16 09:30:30": "[0.31]",
  "2017-06-16 09:30:31": "[0.31]",
  "2017-06-16 09:30:32": "[0.3]",
  "2017-06-16 09:30:33": "[0.29]",
  "2017-06-16 09:30:34": "[0.3]",
  "2017-06-16 09:30:35": "[0.31]",
  "2017-06-16 09:30:37": "[0.29]",
  "2017-06-16 09:30:38": "[0.3]",
  "2017-06-16 09:30:39": "[0.31]",
  "2017-06-16 09:30:40": "[0.27]",
  "2017-06-16 09:30:41": "[0.28]",
  "2017-06-16 09:30:42": "[0.26]",
  "2017-06-16 09:30:43": "[0.27]",
  "2017-06-16 09:30:44": "[0.27]",
  "2017-06-16 09:30:45": "[0.28]",
  "2017-06-16 09:30:46": "[0.37]",
  "2017-06-16 09:30:47": "[0.32]",
  "2017-06-16 09:30:48": "[0.33]",
  "2017-06-16 09:30:49": "[0.34]",
  "2017-06-16 09:30:50": "[0.34]",
  "2017-06-16 09:30:51": "[0.26]",
  "2017-06-16 09:30:52": "[0.33]",
  "2017-06-16 09:30:53": "[0.3]",
  "2017-06-16 09:30:54": "[0.3]",
  "2017-06-16 09:30:55": "[0.29]",
  "2017-06-16 09:30:56": "[0.31]",
  "2017-06-16 09:30:57": "[0.32]",
  "2017-06-16 09:30:58": "[0.28]",
  "2017-06-16 09:30:59": "[0.29]",
  "2017-06-16 09:31:00": "[0.3]"
}
```

Appendix B

Validation Scenarios

B.1 Scenario 1

B.1.1 Selcomps JSON file

```
1
2  {
3    "selcomps": {
4      "GTP_Demo_SelComp": {
5        "ip": "192.168.0.104",
6        "mac": "02-00-4C-4F-4F-50",
7        "port": "8080",
8        "recipes": [],
9        "selcompID": "425",
10       "sensors": {
11         "Bucket Ambient Temperature": "7969",
12         "Bucket Area Humidity": "7968",
13         "Bucket Battery Sensor": "7971",
14         "Bucket Ir Temperature": "7970",
15         "Invisible Light": "7981",
16         "Visible Light": "7980"
17       },
18       "state": "ACTIVE"
19     },
20     "RoboticArm": {
21       "ip": "192.168.1.500",
22       "mac": "65:M5:H8:H8:87",
23       "port": "57681",
24       "recipes": {
25         "115591": {
26           "name": "Shoulder Engine Aquisition Rate",
27           "variables": {
28             "SamplingRate": "Integer"
29           }
```

```
        },
        "115592": {
          "name": "Elbow Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115593": {
          "name": "Wrist Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115594": {
          "name": "Wrist Engine Aquisition Rate",
          "variables": {
            "SamplingRate": "Integer"
          }
        },
        "115595": {
          "name": "ProcessAdjustment",
          "variables": {
            "Accelerometer1": "Integer",
            "Accelerometer2": "Integer",
            "Accelerometer3": "Integer",
            "Timestamp": "Integer"
          }
        },
        "115596": {
          "name": "StopProcess",
          "variables": {
            "Alarm": "Integer"
          }
        }
      },
      "selcompID": "426",
      "sensors": {
        "Base Rotation Engine": "7830",
        "Elbow Engine": "7832",
        "Gripper Engine": "7834",
        "RoboticArm": "7835",
        "RoboticArmStop": "7836",
        "Shoulder Engine": "7831",
        "Wrist Engine": "7833"
      },
      "state": "ACTIVE"
    }
  }
}
```

B.1.2 QR codes

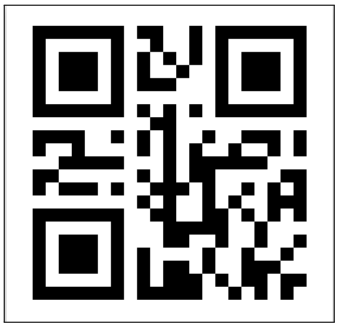

Figure B.1: QR code example ID=426

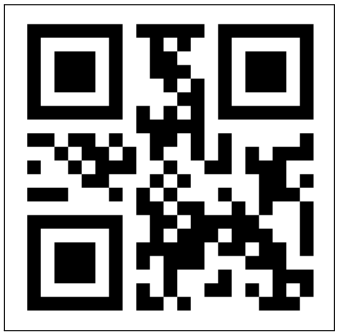

Figure B.2: QR code example ID=425

Appendix C

System interfaces

C.1 Flow diagram

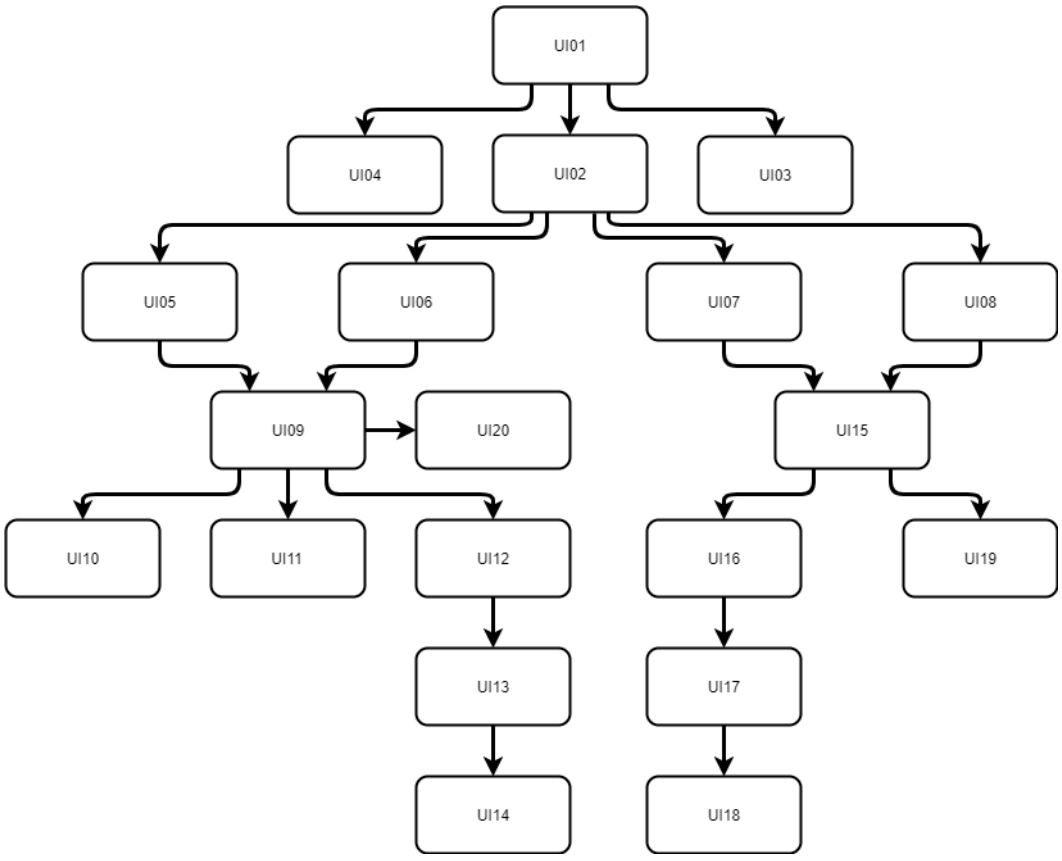

Figure C.1: Application flow diagram

C.2 Interfaces

Figure C.2: UI01 - Initial activity

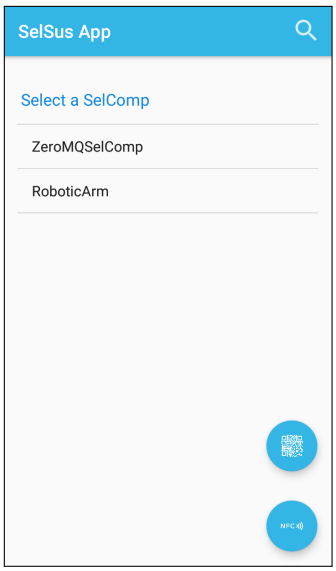

Figure C.3: UI02 - Selcomps list activity

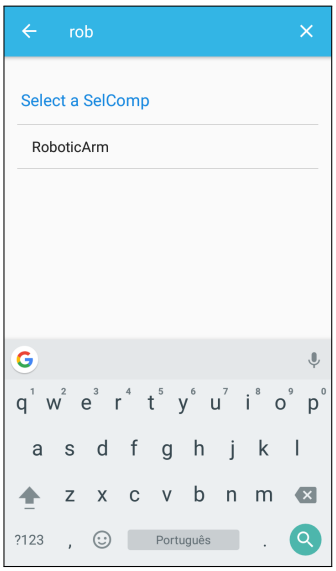

Figure C.4: UI03 - Selcomp search

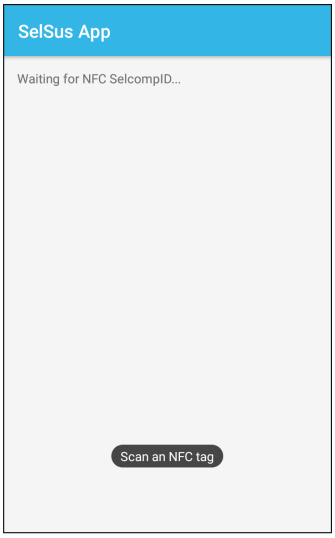

Figure C.5: UI04 - NFC reading activity

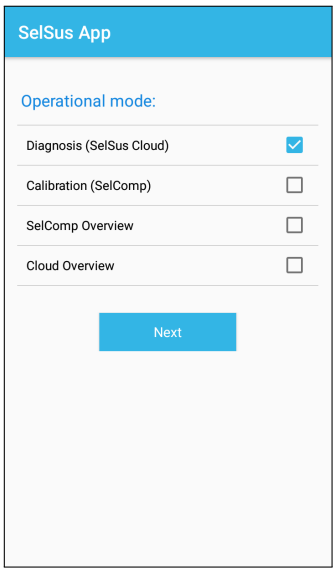

Figure C.6: UI05 - Operational mode 1

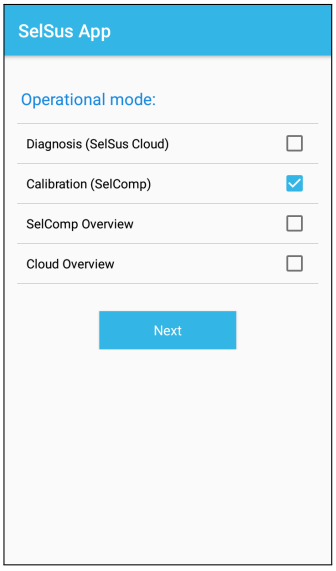

Figure C.7: UI06 - Operational mode 2

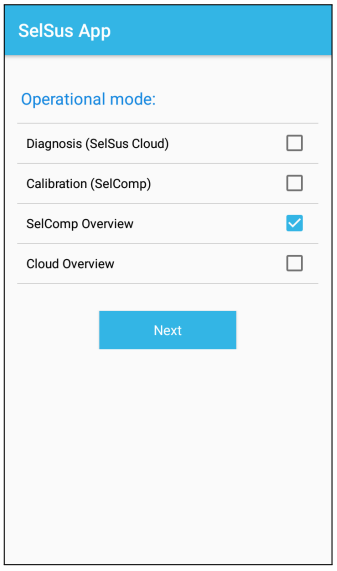

Figure C.8: UI07 - Operational mode 3

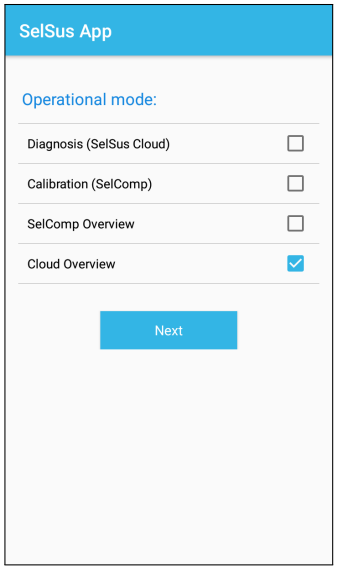

Figure C.9: UI08 - Operational mode 4

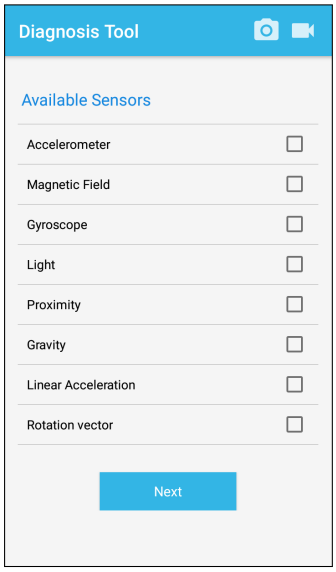

Figure C.10: UI09 - Smartphone embedded sensors list

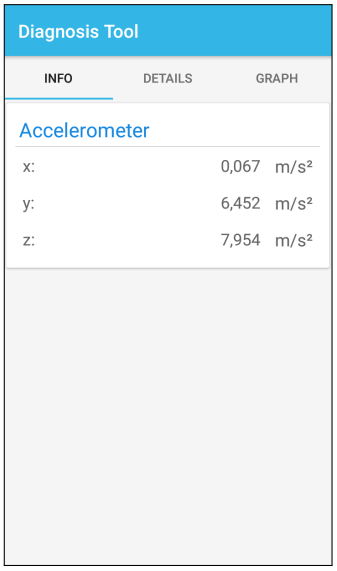

Figure C.11: UI10 - Sensor info tab

Figure C.12: UI11 - Sensor details tab

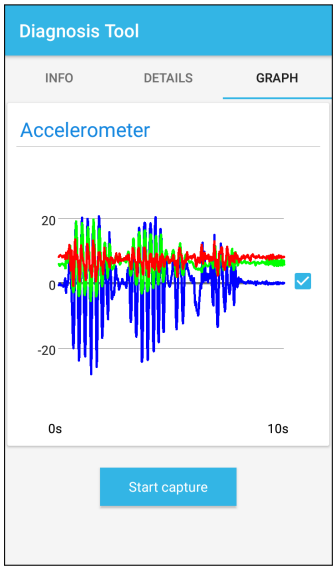

Figure C.13: UI12 - Sensor real-time data tab

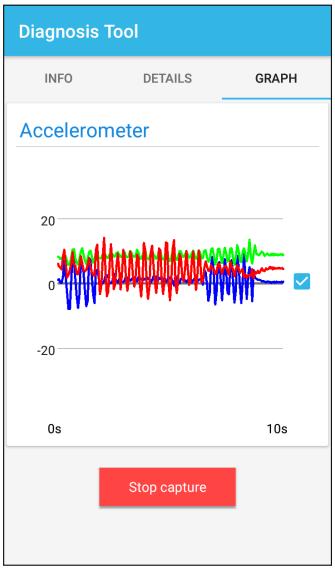

Figure C.14: UI13 - Stop capture

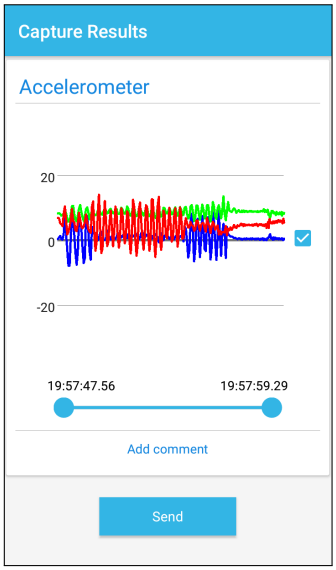

Figure C.15: UI14 - Capture results

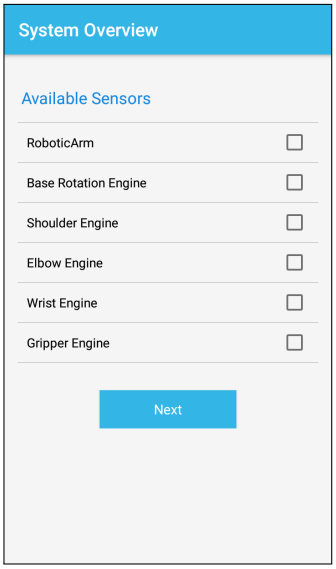

Figure C.16: UI15 - Selcomp sensors activity

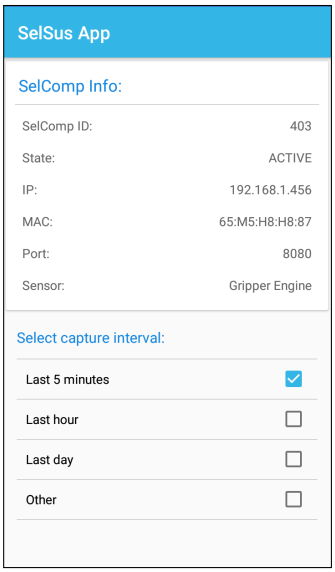

Figure C.17: UI16 - Request parameters

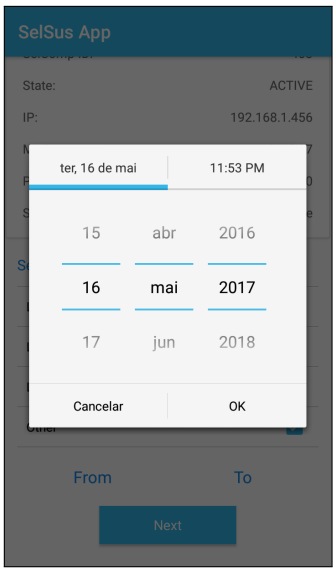

Figure C.18: UI17 - Date and time picker

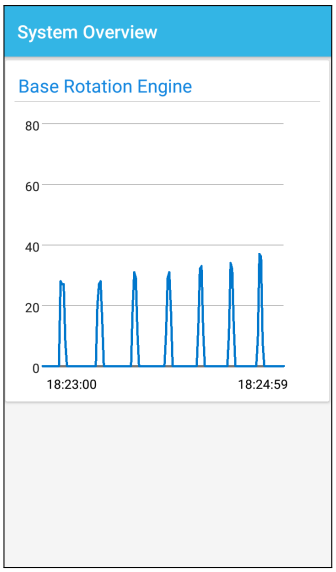

Figure C.19: UI18 - Cloud graph results

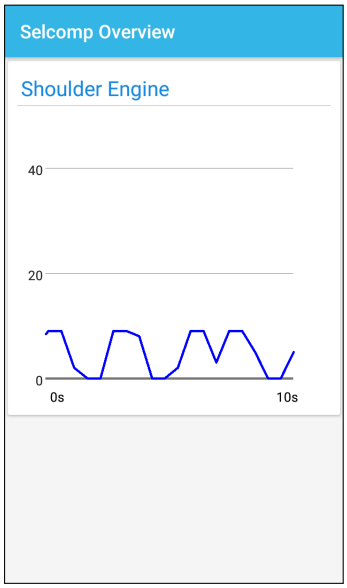

Figure C.20: Selcomp sensor real-time graph

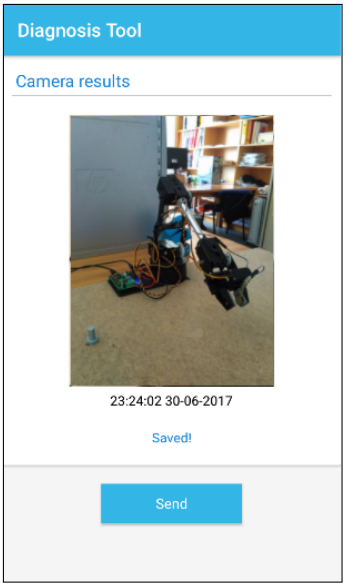

Figure C.21: Media result activity

References

[acc] The Next Industrial Revolution. https://www.accenture.com/us-en/digital-industry-index. Accessed February 7, 2017.

[alt] Prodsmart. http://altizon.com/iot-solutions-for-manufacturing/. Accessed June 13, 2017.

[amqa] Advanced Message Queuing Protocol. https://www.amqp.org/. Accessed February 7, 2017.

[amqb] An Advanced Message Queuing Protocol (AMQP) Walkthrough. https://www.digitalocean.com/community/tutorials/an-advanced-message-queuing-protocol-amqp-walkthrough. Accessed February 7, 2017.

[APG16] Susana Aguiar, Rui Pinto, and Gil Gonc. Life-cycle Approach to Extend Equipment Re-use in Flexible Manufacturing. (c):148–153, 2016.

[BFKR14] Malte Brettel, Niklas Friederichsen, Michael Keller, and Marius Rosenberg. How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective. *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, 8, 2014.

[BG11] Radhakisan Baheti and Helen Gill. Cyber-physical Systems. *The Impact of Control Technology, IEEE*, 2011.

[bos] Industry 4.0 by Bosch: The connected shop floor. http://i40.bosch-si.com/. Accessed February 7, 2017.

[coa] RFC 7252 Constrained Application Protocol. http://coap.technology/. Accessed February 10, 2017.

[dds] DDS The Proven Data Connectivity Standard for the IoT. http://portals.omg.org/dds/. Accessed February 6, 2017.

[FAPG16] Renato Fonseca, Susana Aguiar, Michael Peschl, and Gil Gonc. The ReBorn Marketplace : an Application Store for Industrial Smart Components. (c):136–141, 2016.

[Fer15] Ana Ferreira. A Sensor Cloud Enabler for the Factory of the Future. Master's thesis, Universidade do Porto, 2015.

[Fos] Andrew Foster. A Comparison Between DDS, AMQP, MQTT, JMS, REST and CoAP Version 1.2, November 2013.

[gra] GraphView - open source graph plotting library for Android. http://www.android-graphview.org/. Accessed June 10, 2017.

[HT04] Chi-Fu Huang and Yu-Chee Tseng. The Coverage Problem in a Wireless Sensor Network. *MONET - Mobile Networks and Applications*, 2004.

[ind] Industry 4.0 – fourth industrial revolution. https://blogs.sap.com/2015/06/30/industry-40-fourth-industrial-revolution/. Accessed February 10, 2017.

[Ind13] Industry 4.0 Wroking Group. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. 2013.

[IRA] I-RAMP^3 - Intelligent Network Devices for fast Ramp-up. http://www.i-ramp3.eu. Accessed January 25, 2017.

[jms] Java Message Service Concepts. http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html. Accessed February 10, 2017.

[LML+10] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. AD HOC AND SENSOR NETWORKS A Survey of Mobile Phone Sensing. 2010.

[MDe] Introduction - Material design - Material design guidelines. https://material.io/guidelines/material-design/introduction.html. Accessed June 8, 2017.

[Min11] Roland Minihold. Near Field Communication (NFC) Technology and Measurements White Paper. 2011.

[MMB16] Octavian Morariu, Cristina Morariu, and Theodor Borangiu. Shop-floor resource virtualization layer with private cloud support. *Journal of Intelligent Manufacturing*, 27(2):447–462, apr 2016.

[Mon14] László Monostori. ScienceDirect Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia CIRP*, 17:9–13, 2014.

[MP16] George Manea and Sorin Popa. INTEGRATION OF SENSOR NETWORKS IN CLOUD COMPUTING. *U.P.B. Sci. Bull., Series C*, 78(2), 2016.

[MQTa] MQTT - Frequently Asked Questions. http://mqtt.org/faq. Accessed June 10, 2017.

[MQTb] MQTT 101 – How to Get Started with the lightweight IoT Protocol. http://www.hivemq.com/blog/how-to-get-started-with-mqtt. Accessed June 10, 2017.

[nod] Variable's NODE+ solution. http://www.variableinc.com/node1. Accessed June 8, 2017.

[pro] Prodsmart. https://prodsmart.com/pt-pt/. Accessed February 10, 2017.

[PRS+15] Rui Pinto, João Reis, Ricardo Silva, Vitor Sousa, and Gil Gonçalves. Self-organising Smart Components in Advanced Manufacturing Systems. pages 157–163, 2015.

REFERENCES

[PRS+16] R Pinto, J Reis, R Silva, M Peschl, and G Gonçalves. Smart Sensing Components in Advanced Manufacturing Systems. *International Journal on Advances in Intelligent Systems*, 9(1 & 2), 2016.

[quaa] Qualcomm. https://www.qualcomm.com/. Accessed June 8, 2017.

[quab] Smartphone sensor growth. https://www.qualcomm.com/news/onq/2014/04/24/behind-sixth-sense-smartphones-snapdragon-processor-sensor-engin Accessed June 8, 2017.

[ReB] ReBorn Innovative Reuse. http://www.reborn-eu-project.org/about.html. Accessed January 25, 2017.

[RJS16] Griessbauer Reinhard, Vedso Jesper, and Schrauf Stefan. Industry 4.0: Building the digital enterprise. *2016 Global Industry 4.0 Survey*, pages 1–39, 2016.

[RPG17] João Reis, Rui Pinto, and Gil Gonc. Human-Centered Application using Cyber-Physical Production System. 2017.

[sam] Samsung. http://www.samsung.com/global/galaxy/galaxy-s8/specs/. Accessed June 8, 2017.

[Sel] SelSus. http://selsus.eu. Accessed January 25, 2017.

[sena] Sensorbug solution. http://www.blehome.com/sensorbug-lp.html. Accessed June 8, 2017.

[senb] Sensors Overview | Android Developers. https://developer.android.com/guide/topics/sensors/sensors{_}overview.html. Accessed June 8, 2017.

[sma] Gartner Says Tablet Sales Continue to Be Slow in 2015. https://www.gartner.com/newsroom/id/2954317. Accessed February 6, 2017.

[Soo08] Tan Jin Soon. QR Code. *Synthesis journal*, 2008.

[SS07] Stonebraker and Seltzer. Linux Security Sins Standards-Based Middleware API Design Matters. *Queue May*, 5(4), 2007.

[staa] Forecast installed base of NFC-enabled phones worldwide from 2013 to 2018 (in millions). https://www.statista.com/statistics/347315/nfc-enabled-phone-installed-base. Accessed February 10, 2017.

[stab] Smartphone users worldwide 2014-2020. www.statista.com/statistics/330695/number-of-smartphone-users-worldwide. Accessed February 6, 2017.

[tul] Tulip, the App Engine for Shopfloors. https://tulip.co/. Accessed February 7, 2017.

[ver] Vernier sensor solutions. https://www.vernier.com/products/sensors/. Accessed June 8, 2017.

[WADX15] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, Apr 2015.

[whi] White paper on sensor cloud. Technical Report Deliverable 3.5, Institute of Systems and Robotics, Faculty of Engineering, University of Porto.

[xcu] [x]cube Labs. https://www.xcubelabs.com/. Accessed February 7, 2017.

[XPR] XPRESS - Steinbeis Europa Zentrum. https://www.steinbeis-europa.de/en/sectors-projects/advanced-manufacturing-technologies/xpress.html. Accessed January 25, 2017.